

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 07-12-2014		2. REPORT TYPE Final Technical Report		3. DATES COVERED (From - To) Apr. 01, 09 - Sept. 30, 14	
4. TITLE AND SUBTITLE Computational Modeling and High Performance Computing in Advanced Material Processing, Synthesis, and Design				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER N00014-09-1-0842	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Ram Mohan				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) North Carolina A and T State University 1601 E Market Street Greensboro, NC 27411-0001				8. PERFORMING ORGANIZATION REPORT NUMBER Final Report 210111	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 875 North Randolph Street Arlington, VA 22203-1995				10. SPONSOR/MONITOR'S ACRONYM(S) ONR 332, Dr. Kenny Lipkowitz	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited					
13. SUPPLEMENTARY NOTES The views, opinions, and/or findings contained in this report are those of the author(s).					
14. ABSTRACT Computational modeling and simulations involving STEM (science, technology, engineering and mathematics) disciplines are highly interdisciplinary growing out of complex, challenging, multi-domain, multi-component and data intensive application needs of several disciplines and the need for high end computing hardware and software, algorithms and information driven technologies. The research efforts in this project focused on the synergistic coupling of: <ul style="list-style-type: none"> • Computational material science and mechanics of hybrid and light weight polymeric composite structures • Computational multi-scale deformation behavior in metallic material systems • Computational enabling technologies of multi-scale flow simulation investigations with Lattice-Boltzmann equations and finite volume methods; physics based composite process flow modeling in GPU systems; and probabilistic methods to understand uncertainties in deterministic composite processing flow models. 					
15. SUBJECT TERMS Computational modeling, high performance computing, material science and mechanics, polymer composites, metallic nanolayered composites, nanomechanics, probabilistic methods					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 320	19a. NAME OF RESPONSIBLE PERSON Ram Mohan
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER (include area code) 336-285-2867

Award Information

FINAL TECHNICAL REPORT	
Award Number	N00014-09-1-0842
Title of Research	Computational Modeling and High Performance Computing in Advanced Material Processing, Synthesis, and Design
Principal Investigator	Dr. Ram Mohan
Organization	North Carolina A&T State University, Greensboro, NC
Period of Performance	April 1, 2009 – September 30, 2014

Final Technical Report
April 01, 2009 – September 30, 2014
ONR Award Number: N00014-09-1-842
PR Number: 09PR07107-00

**Computational Modeling and High Performance Computing in Advanced
Materials Processing, Synthesis, and Design**

Performing Institution
North Carolina A&T State University (PI: Prof. Ram Mohan)

Office of Naval Research Technical Representative
Dr. Kenny Lipkowitz (Code: ONR 332)

Technical Point of Contact

Prof. Ram Mohan
Professor, Nanoengineering
Joint School of Nanoscience and Nanoengineering
North Carolina A&T State University
2907 E Lee Street
Greensboro, NC 27401
Phone: 336-285-2867
Fax: 336-500-0115
E-mail: rvmohan@ncat.edu

20141210041

TECHNICAL SECTION	4
TECHNICAL OBJECTIVES	4
TECHNICAL APPROACH	7
TECHNICAL ACCOMPLISHMENTS SUMMARY	11
A-1: COMPUTATIONAL MATERIAL SCIENCE AND MECHANICS OF HYBRID AND LIGHT WEIGHT POLYMERIC COMPOSITE STRUCTURES	11
A-1-1: <i>Atomistic Modeling in Polymer Nanocomposite Systems – Applications to Mechanics of Multi-Scale Materials</i>	11
A-1-2: <i>Modeling and Experimental Investigation on the Effect of Interlaminar Nanofiber Layers on the Delamination Behavior in an Epoxy Fiber Glass Composite</i>	12
A-2 COMPUTATIONAL MULTI-SCALE DEFORMATION BEHAVIOR IN METALLIC MATERIAL SYSTEMS	12
A-2-1 <i>Molecular Dynamics Modeling of tensile, flexural and crack propagation in metallic systems</i>	13
A-2-2 <i>Mechanical Behavior of Nanoscale Metallic Composites – Dynamic Crack Propagation in Ni-Al Bilayer Composite</i>	13
B-1 MULTI-SCALE SIMULATION INVESTIGATIONS OF NANOFIBER RESIN INTERACTIONS USING LATTICE BOLTZMANN EQUATIONS AND FINITE VOLUME METHODS.....	14
B-2 PHYSICS BASED MODELING AND SIMULATION ON GRAPHICAL PROCESSING UNITS (GPUS) – POROUS MEDIA FLOW IN LIQUID COMPOSITE MOLDING	14
B-3 PROBABILISTIC ANALYSIS OF PROPERTY UNCERTAINTIES USING RESIN INFUSION FLOW MODELING AND SIMULATIONS.....	15
DETAILED TECHNICAL REPORT	16
A-1: COMPUTATIONAL MATERIAL SCIENCE AND MECHANICS OF HYBRID AND LIGHT WEIGHT POLYMERIC COMPOSITE STRUCTURES	16
A-1-1: <i>Atomistic Modeling in Polymer Nanocomposite Systems – Applications to Mechanics of Multi-Scale Materials</i>	16
Computational Study of the Effect of Carbon Vacancy Defects on the Young's Modulus of (6,6) Single Wall Carbon Nanotube	16
Predictive Mechanical Properties of EPON 862 (DGEBF) cross-linked with Curing Agent W (DETDA) and SWCNT using MD Simulations – Effect of Carbon Vacancy Defects	34
A-1-2: <i>Modeling and Experimental Investigation on the Effect of Interlaminar Nanofiber Layers on the Delamination Behavior in an Epoxy Fiber Glass Composite</i>	49
A-2 COMPUTATIONAL MULTI-SCALE DEFORMATION BEHAVIOR IN METALLIC AND NON-METALLIC SYSTEMS.....	62
A-2-1 <i>Molecular Dynamics Modeling of tensile, flexural and crack propagation in metallic systems</i>	62
A-2-2 <i>Mechanical Behavior of Nanoscale Metallic Composites – Dynamic Crack Propagation in Ni-Al Bilayer Composite</i>	88
B-1 MULTI-SCALE SIMULATION INVESTIGATIONS OF NANOFIBER RESIN INTERACTIONS USING LATTICE BOLTZMANN EQUATIONS AND FINITE VOLUME METHODS.....	109
B-2 PHYSICS BASED MODELING AND SIMULATION ON GRAPHICAL PROCESSING UNITS (GPUS) – POROUS MEDIA FLOW IN LIQUID COMPOSITE MOLDING	144
<i>Architecture–Performance Interrelationship Analysis in Single/Multiple CPU/GPU Computing Systems: Application to Composite Process Flow Modeling</i>	144
B-3 PROBABILISTIC ANALYSIS OF PROPERTY UNCERTAINTIES USING RESIN INFUSION FLOW MODELING AND SIMULATIONS.....	286
<i>Preform and Resin property uncertainties, role, and their effect in liquid composite process flow modeling..</i>	286
<i>Statistical Analysis of Uncertainties in Deterministic Computational Modeling – Application to Composite Process Resin Infusion Flow Modeling</i>	301
C. EDUCATIONAL ACTIVITY AND STUDENT SUPPORT	317
D. COMPUTATIONAL AND VISUALIZATION HARDWARE / SOFTWARE	320

Technical Section

Technical Objectives

Computational modeling and simulations involving STEM (science, technology, engineering and mathematics) disciplines are highly interdisciplinary growing out of complex, challenging, multi-domain, multi-component and data intensive application needs of several disciplines and the need for high end computing hardware and software, algorithms and information driven technologies. The research efforts in this project focused on the synergistic coupling of:

- A. Research relevant to Navy/DOD in computational sciences
- B. Computational Simulation, Visualization and Enabling Technologies
- C. Student support towards development of next generation workforce
- D. Associated support of computational and visualization hardware/software needs

A. Research Relevant to Navy/DOD in Computational Sciences

The Navy/DOD relevant research activities emphasize on the computational science research with a focus on a paradigm for computer assisted material design and validation for material systems that include metals, polymers and composites.

New material design developments and interface between different constituent materials can be understood by studying the fundamental interactions that exist across the functional material element constituents. Such fundamental understanding can be enabled by materials design through modeling and simulation of materials and material interfaces envisioned in multi-functional and hybrid material systems. The active, optical, biological, structural, and/or electronic simulations of different materials are required to achieve successful scientific breakthroughs in the development and design of new class of material systems. These modeling, simulation and visualization techniques are critical for the specific understanding of the interface and interactions between the organic and inorganic material systems, in particular the complex multi-length and temporal scale material and multi-species interactions that will enable and optimize the development, processing, fabrication and scaling of the heterogeneous material systems. Physical, computational modeling and simulations are still lagging for such heterogeneous, multi-scale (length and time), multi-component interface material systems and formed the research focus of this project.

The proposed relevant research activities under this umbrella are defined and categorized under these focus areas. They are:

A-1: Computational Multi-Scale Science and Mechanics for Hybrid Light Weight Polymeric Composite Structures

A-2: Computational Multi-scale Deformation Behavior in Metallic Material Systems

B. Computational Simulation, Visualization and Enabling Technologies

Several enabling technologies, their research development and availability are critical to the success of the research and education activities in computational modeling and high performance computing in advanced materials processing and synthesis and design. Toward this end, the proposed efforts involve the development of an expanded computational simulation and enabling technology and the research in the associated enabling technology areas including applications of new high performance computing paradigms for physics based modeling and simulations.

C. Educational Activity and Student Support

North Carolina A & T State University (NCAT) is a land grant, historically black college and university (HBCU) with the graduate Masters Computational Science and Engineering (CSE) program established in 2005 and Ph.D. program in CSE established in 2010. NCAT has also now established the graduate programs (MS and Ph.D.) in nanoengineering. Research initiated and established through the ONR award provided the foundation for this establishment of the new nanoengineering graduate program and paved the way for NCAT to lead such efforts in this arena. NCAT is the first HBCU with a graduate master's and doctoral program in nanoengineering. The educational activities of the proposed efforts were instrumental in supporting the educational infrastructure requirements of the current Master's and Ph.D. programs in nanoengineering and CSE. The educational activity funds of the project provided financial fellowships and assistantships to the Ph.D. and Master's students in the area of computational mechanics, nanomechanics, material sciences, and enabling technologies. This is enabling the education and training of future workforce (esp. underrepresented minorities) in these critical technology areas. In addition, North Carolina A&T State University has initiated a new graduate program in Nanoengineering with computational nanoengineering as one of the focus areas. The research areas related to computational nanomechanics, multi-scale modeling and nanoengineered materials leveraged and enabled through the ONR funding benefited the students of this program and leveraged additional opportunities from this new graduate program.

D. Computational and Visualization Hardware / Software

Computational and visualization hardware/software resources are critical components of computational modeling research. The project funding was instrumental in supporting the associated software and system upgrades benefiting the research and educational needs of the students, faculty and the research focus. In addition, the project funding was instrumental in the award of a NSF major research instrumentation award to acquire a multi-processor SUN Blade system that is currently under installation. ONR project funds in part were also used to support the acquisition of a new multi-processor Cray XC-30 system that is now providing the hardware needs for computational nanoengineering at North Carolina A&T State University.

Technical Approach

All the research and educational developments and investigations performed under this grant award targeted towards advancing the state of the art in the computational science and modeling approaches for the class of problems and applications with multiple length and time scales and that would require bridging across the different scales. The exploratory research developments provided and enhanced the understanding of computational modeling and simulation technologies towards a material by design paradigm for naval material developments, systems, and applications.

Based on the above research, education, technology focus, specific research investigations were initiated and conducted during the project funding period. Brief details of these research investigations during the project period and technical approaches are presented here. Further technical discussions are reported in the section on “Detailed Technical Discussions”. All these research investigations have resulted and are resulting in peer-reviewed publications, post-doctoral research training, graduate student education and graduation through thesis and doctoral dissertations associated with these research investigations.

A-1: Computational material science and mechanics of hybrid and light weight polymeric composite structures

Research activities and the associated technical approach in the area A-1 conducted during the project period focused on:

- Molecular dynamics modeling of carbon nano tube (CNT) – epoxy composites.
- Modeling and experimental investigation on the effect of Intelaminar Nanofiber layers on the deformation behavior.

A-1-1: Atomistic Modeling in Polymer Nanocomposite Systems – Applications to Mechanics of Multi-Scale Materials

Hybrid and nano composites are formed with material phases at varying length scales, and include nano material constituents. The behavior of these composites is influenced by the material interactions during processing, and by the damages/defects in the associated constituent nano materials. Low length scale modeling based on the atomistic, molecular structures provides an insight into the molecular level interactions that exist, and their influence on the associated composite properties. Such modeling can also provide predictive properties and an understanding

of the atomistic level behaviour. The project efforts focused on the role of atomistic modelling and simulations with a focus on applications to a hierarchical nanoengineered composite material systems consisting of carbon nanotube epoxy nanocomposites.

A-1-2: Modeling and Experimental Investigation on the Effect of Interlaminar Nanofiber Layers on the Delamination Behavior in an Epoxy Fiber Glass Composite

This work focused on the addition of electrospun nano fiber interface layers between the traditional composite laminates and its effect on the delamination characteristics in an epoxy-fiber glass composite system. Electrospun glass nano fiber layers formed with TEOS (Tetra Ethyl Ortho Silicate) sol gel system are used as interface layers. Delamination characteristics of the composite with and without electrospun fiber interface layers are studied using double cantilever beam (DCB) tests. The experimental characterization showed that addition of nano fiber layers provided consistent improvements in the Mode-I fracture toughness values. Finite element modeling of the crack growth and delamination failure with and without the nano fiber layers are studied and compared. The Mode-I fracture toughness values from the finite element modeling are compared with the experimental data.

A-2 Computational multi-scale deformation behavior in metallic material systems

Research activities and technical approach in this area during the project period focused on the deformation behavior of nanoscale material systems with applications to tensile, flexural, and crack propagation.

A-2-1 Molecular Dynamics Modeling of tensile, flexural and crack propagation in metallic systems

Nanomechanics is an evolving field that investigates the mechanical properties, deformation behavior and characteristics of nanoscale structures. Due to the smaller lengths at the nano level, principles of mechanics are employed in conjunction with interatomic potentials, molecular forces and molecular dynamics. This work focused on the tensile and flexural deformation of Nickel nanowires; and dynamic crack propagation in nanoscale Nickel and Nickel-Aluminum bimetal interface.

A-2-1 Mechanical Behavior of Nanoscale Metallic Composites – Dynamic Crack Propagation in Ni-Al Bilayer Composite

Nanoscale multilayer metallic composites (NMMCs) contain significantly high volume fraction of interfaces and exhibit strengths much higher than that of bulk materials composing the structures. This strengthening has been attributed to the presence of interfaces between materials

that differ in properties such as elastic modulus, lattice parameters, slip plane orientations and act as barriers to propagating dislocations. This work focused on the influence of semi-coherent Ni (nickel) - Al (aluminum) interface on Mode-I crack propagation in nanoscale Ni-Al bilayer composite under tensile and cyclic loading conditions analyzed through computational modeling.

B: Computational enabling technologies

The specific research activity and technical approach in area B are:

B-1 Multi-Scale Simulation Investigations of Nanofiber Resin Interactions using Lattice Boltzmann Equations and Finite Volume Methods

Multi-scale modeling approaches are required to accurately capture the disparate length scale effects in various engineering problems. Preliminary work in this area focused on the coupled Lattice Boltzmann and Navier Stokes modeling for flow problems in collaboration with University of Alabama at Birmingham. Further developments in these concurrent coupled modeling developments are needed. The present efforts are geared towards applications in understanding the nano fiber, nano tube resin flow interactions in composites material processing. Due the low length scale size of nanofibers in comparison to the resin flow domain, low length scale methods in the vicinity of the nanofiber flow region and correlation with the macroscopic flow field. The present research investigations and modeling investigations comparing the Lattice Boltzmann and Navier Stokes approaches that are in progress are presented.

B-2 Physics Based Modeling and Simulation on Graphical Processing Units (GPUs) – Porous Media Flow in Liquid Composite Molding

High performance computing architectures are evolving over the years with the Graphical Processing Units (GPU) are providing superior performances for computationally intensive problems. Results from the continued investigations are presented.

B-3 Probabilistic Analysis of Property Uncertainties using Resin Infusion Flow Modeling and Simulations

Physics based flow modeling provides an effective way to simulate the resin infusion process in liquid composite molding processes for polymer composite structures. These are effective to provide optimal injection time and locations for given process parameters of resin viscosity and preform permeability prior to resin gelation. However, there could be significant variations in these two parameters during actual manufacturing due to differences in the resin batches, mixes,

temperature, ambient conditions for viscosity; in the preform rolls, compaction, etc., for permeability. Research to understand the influence of uncertainties in these parameters on the resin infusion time was initiated via a probabilistic modeling methodology using resin flow modeling and statistical analysis.

All these research investigations have resulted and are resulting in peer-reviewed publications, post-doctoral research training, graduate student education, continuing graduate work and graduation.

Technical Accomplishments Summary

New material design developments require a fundamental understanding through computational modeling of materials, interfaces and associated mechanics. Project efforts demonstrated the effectiveness of computational modeling in material processing, synthesis and design via: Molecular Dynamics modeling (MD) of carbon vacancy defects as a potential cause to reduce modulus of SWNCT, and SWCNT-epoxy composites; Molecular Dynamics modeling of deformation and Ni-Al bimetal interface for insight on nanoscale fracture; Lattice Boltzmann and Finite Volume Method for coupled meso- macro- flow analysis; Graphical processing unit (GPU) as a computing platform for composite process flow modeling; Non-deterministic probabilistic based methods to understand the influence of processing parameters based on deterministic models.

A summary of the research and technical accomplishments and results obtained during the project period in the various research investigations are presented next.

A-1: Computational material science and mechanics of hybrid and light weight polymeric composite structures

Research activities and the associated technical approach in the area A-1 conducted during the project period focused on:

- Molecular dynamics modeling of carbon nano tube (CNT) – epoxy composites.
- Modeling and experimental investigation on the effect of Intelaminar Nanofiber layers on the deformation behavior.

A-1-1: Atomistic Modeling in Polymer Nanocomposite Systems – Applications to Mechanics of Multi-Scale Materials

Computational techniques such as molecular dynamics (MD) simulations have emerged as an alternative to the traditional experimental and theoretical methods of estimating mechanical properties of the Epoxy – Carbon Nanotube composite systems. However, differences have been observed between results obtained from experiments and those obtained from MD simulations with the experimental results being lower. The effect of carbon vacancy defects in the single wall carbon nanotube (SWCNT) on the Young's modulus of the nanotubes as well as their EPON 862-

DETDA-SWCNT composite evaluated through molecular dynamics simulations performed with Accelrys and Materials Studio were focused in the present work. Since their discovery, Carbon nanotubes (CNT) have gained significant attention because of their superior chemical, mechanical and thermo-physical properties. Inclusion of CNTs in polymer matrices have shown significant improvement of properties compared with the properties of the parent polymers, however, defects in these CNTs have also been observed to have detrimental effects on the mechanical properties of the composites.

A-1-2: Modeling and Experimental Investigation on the Effect of Interlaminar Nanofiber Layers on the Delamination Behavior in an Epoxy Fiber Glass Composite

Delamination is one of the important failure mechanisms in composite materials. Several methods such as stitching of fiber plies, self-healing polymer materials, and interface reinforcements have been developed, investigated and employed over the years to improve the delamination characteristics. The usage of interface material layers (in particular, sub-micron and nano level materials) has also been recently investigated. This study focused on the addition of electrospun nano fiber interface layers between the traditional composite laminates and its effect on the delamination characteristics in an epoxy-fiber glass composite system. Electrospun glass nano fiber layers formed with TEOS (Tetra Ethyl Ortho Silicate) sol gel system are used as interface layers. Delamination characteristics of the composite with and without electrospun fiber interface layers are studied using double cantilever beam (DCB) tests. The experimental characterization showed that addition of nano fiber layers provided consistent improvements in the Mode-I fracture toughness values. Finite element modeling of the crack growth and delamination failure with and without the nano fiber layers are studied and compared. The Mode-I fracture toughness values from the finite element modeling are compared with the experimental data.

A-2 Computational multi-scale deformation behavior in metallic material systems

Research activities and technical approach in this area during the project period focused on the deformation behavior of nanoscale material systems with applications to tensile, flexural, and crack propagation.

A-2-1 Molecular Dynamics Modeling of tensile, flexural and crack propagation in metallic systems

Nanomechanics is an evolving field that investigates the mechanical properties, deformation behavior and characteristics of nanoscale structures. Due to the smaller lengths at the nano level, principles of mechanics are employed in conjunction with interatomic potentials, molecular forces and molecular dynamics. This work focused on the tensile and flexural deformation of Nickel nanowires; and dynamic crack propagation in nanoscale Nickel and Nickel-Aluminum bimetal interface.

The tensile deformation behavior analysis indicates that Young's Modulus was independent of the cross sectional area of the nanowire, and the strain rate. The flexural deformation and vibration behavior indicates that the frequency of the vibrations as computed from time displacement deformation behavior of the molecular configurations of the Nickel nanowire beams are independent of the magnitude of external loading, and is consistent with the classical beam theory.

The dynamic crack propagation behavior in a Nickel single crystal and a Nickel-Aluminum bimetal interface are investigated. The propagation mechanisms and fracture behavior in Ni are compared with such behavior in Ni-Al nanoscale bimetallic layer that initiates and propagates from Ni towards the Ni-Al bimetal interface. Our results for Ni show an initial brittle crack propagation followed by a roughening of the crack surfaces at one-third of the Rayleigh wave speed. In Ni-Al, the crack surfaces initially grow brittle. Two regimes of crack propagation velocities were observed in this case with crack getting decelerated as it nears the interface. Further dynamic analysis of the crack propagation indicated a cease in the crack propagation in Ni due to a brittle to ductile transition. In Ni-Al bimetal interface system, as the crack approaches the interface, a process zone representing local disorder at the crack tip was observed to start growing and interacting with interfacial defects that eventually results in a blunting of the crack tip.

A-2-2 Mechanical Behavior of Nanoscale Metallic Composites – Dynamic Crack Propagation in Ni-Al Bilayer Composite

Nanoscale multilayer metallic composites (NMMCs) contain significantly high volume fraction of interfaces and exhibit strengths much higher than that of bulk materials composing the structures. This strengthening has been attributed to the presence of interfaces between materials that differ in properties such as elastic modulus, lattice parameters, slip plane orientations and act as barriers to propagating dislocations. This paper presents a review of two major factors that influence the properties and behavior of the NMMCs: Interface structure,

Strengthening/Deformation mechanisms. The influence of semi-coherent Ni (nickel) - Al (aluminum) interface on Mode-I crack propagation in nanoscale Ni-Al bilayer composite under tensile and cyclic loading conditions analyzed through computational modeling is discussed. Results for nanoscale Ni-Al bilayer composite showed initial brittle crack propagation with planar cleavage of atoms followed by crack surfaces getting roughened when crack propagation speed is about one-third of Rayleigh wave speed. In case of Mode-I tensile cyclic loading, crack was found to propagate either by fatigue cleavage of the atoms or by void nucleation in the regions near the crack tip, depending on the value of maximum strain applied. In Ni-Al bilayer composite studied, as crack approached the interface, dislocations start emanating from the interfacial layer. The creation of voids was found to slow down crack growth in both the Ni and Ni-Al at higher maximum applied strain during cyclic loading. Plastic deformation was found to dominate crack propagation during tensile loading that resulted in a slower crack growth than cyclic loading. In all cases, presence of semi-coherent interface in the nanoscale Ni-Al bilayer composite was found to prohibit crack from propagating beyond the interface.

B-1 Multi-Scale Simulation Investigations of Nanofiber Resin Interactions using Lattice Boltzmann Equations and Finite Volume Methods

The orientation/distribution of carbon nanotube (CNT) and other nanofibers in polymer matrix, one of main factors in manufacturing high-performance multifunctional composites, is an important aspect to be considered during the development of new CNT composites with enhanced mechanical, electrical and thermal properties. However, the disparate length scales involved and mechanical properties of nanotube and rheological properties of polymer matrix around CNT and nanofibers hinder researchers from elucidating the problem via computational modeling. Understanding this problem requires a multi-scale computational approach and the associated enabling technologies. Different computational solvers for each of these scales, bridging techniques between the solvers, and a representative model of a carbon nanotube/nanofiber are needed for the simulation of this class of multi-scale and multi-disciplinary problems. The activities, accomplishments and results during the past year focused on the 1) the coupling of a macro-scale solver, HYB3D, and a meso-scale solver, Regularized Lattice Boltzmann (LB) equation solver, for computational fluid dynamics problems.

B-2 Physics Based Modeling and Simulation on Graphical Processing Units (GPUs) – Porous Media Flow in Liquid Composite Molding

The present year efforts examined the performance of graphics hardware when used as a co-processor within the context of a real-world application. The real-world application is Resin

Flow Infusion using the Finite Element Method (FEM) as provided by pre-existing Liquid Composite Molding (LCM) software. As illustrated in this study, the Graphics Processing Unit (GPU) used as a co-processor provides a definite boost in performance. However, the inherent differences in the GPU and CPU paradigms necessitate a different software structure.

B-3 Probabilistic Analysis of Property Uncertainties using Resin Infusion Flow Modeling and Simulations

Physics based flow modeling provides an effective way to simulate the resin infusion process in liquid composite molding processes for polymer composite structures. These are effective to provide optimal injection time and locations for given process parameters of resin viscosity and preform permeability prior to resin gelation. However, there could be significant variations in these two parameters during actual manufacturing due to differences in the resin batches, mixes, temperature, ambient conditions for viscosity; in the preform rolls, compaction, etc., for permeability. Research to understand the influence of uncertainties in these parameters on the resin infusion time was initiated via a probabilistic modeling methodology using resin flow modeling and statistical analysis.

In addition to the research progress, the present project efforts in the establishment of the MS and Ph.D. program in nanoengineering and in developing the computational nanoengineering focus areas. The project initiatives were instrumental in providing the leverage for new research projects in the areas of multi-scale modeling of cementitious materials, high performance computational modeling of bio-nano interfaces. These would not have been possible without the enabling support provided by this research funding. The project efforts are benefiting the graduate student research education and training for the students in CSE and the nanoengineering programs. Several of these students participated in the research activities for the past year and continue to participate in the research activities. The research areas related to computational nanomechanics, multi-scale modeling and nanoengineered materials leveraged and enabled through the present funding is benefiting the students of this program and in facilitating new research opportunities in the computational nanoengineering, and attracting qualified minorities to the research areas of computational nanoengineering.

Detailed Technical Report

Detailed technical report of the methodology, results and discussions are presented next. Technical discussions are organized under the same categories and headings as listed in the “Technical Approach” section. In addition, the accomplishments from the educational activity and student support and computational hardware/software are also presented.

A-1: Computational material science and mechanics of hybrid and light weight polymeric composite structures

A-1-1: Atomistic Modeling in Polymer Nanocomposite Systems – Applications to Mechanics of Multi-Scale Materials

Computational Study of the Effect of Carbon Vacancy Defects on the Young’s Modulus of (6,6) Single Wall Carbon Nanotube

Authors: E. Fefey, R. Mohan, A. Kelkar, North Carolina A&T State University

Published Journal Article: *Materials Science and Engineering: B*, Vol. 176, Issue 9, Pages 693-772, 2011(doi:10.1016/j.mseb.2011.02.019)

Abstract

Computational techniques such as molecular dynamics (MD) simulations have emerged as an alternative to the traditional experimental and theoretical methods of estimating mechanical properties of single wall carbon nanotubes (SWCNTs) and polymer nanocomposites containing SWCNTs. Most MD simulations are based on a perfect molecular material structure of the SWCNT. The presence of vacancy defects in SWCNTs could lead to deviations from this perfect structure thus affecting the predicted properties. The present study investigated the effect of carbon vacancy defects in the molecular structure of SWCNT on the Young’s modulus of the SWCNT using MD simulations performed via Accelrys and Materials Studio. The effect of the position of the defects in the nanotube ring and the effect of the number of defects on the Young’s modulus are studied. The studies indicate that for an enclosed defect with the same

shape in a SWCNT structure, its position did not cause any change in the Young's modulus. However, as the number of defects increased, the predicted Young's modulus was found to decrease. For a 10 ring (6, 6) SWCNT, six vacancy defects (corresponding to a defect percentage of 2.5%) reduced the Young's modulus by 13.7%. These results indicate that presence of carbon vacancy defects are one potential cause for the reduction and lower Young's modulus of SWCNT, and subsequently lower Young's Modulus obtained experimentally in SWCNT dispersed epoxy-SWCNT nanocomposites cited in the literature.

1. Introduction

Since their discovery in 1991 [1], single wall carbon nanotubes (SWCNTs) have gained significant attention because of their superior chemical, mechanical and thermo-physical properties. For example, in the field of polymer nanocomposites, inclusion of SWCNTs into polymer matrices have been reported to result in significant improvement of properties compared to the properties of the parent polymers [2, 3]. The mechanical properties of SWCNTs and that of polymer nanocomposites containing SWCNTs have been studied through theoretical, experimental and computational analysis [4-7]. Results reported in the literature have indicated that mechanical properties obtained from experiments tend to be lower than those obtained from the computational analysis [8]. Several reasons have been assigned for this disparity including the fact that the experimental macroscopic coupons have SWCNTs dispersed in various orientations, while the computational models usually have the SWCNTs uni-directionally aligned. Another cause for this disparity is the fact that experimental processes could introduce defects into the SWCNTs, while computational models are based on a perfect SWCNT molecular material structure. Carbon vacancy defects in the molecular structure have been known to cause changes in the properties of CNTs [9-12]. Chemically, in some cases, these defects have been reported to enhance the affinity of the CNTs at the defective site making them most suitable as Platinum carrier electrodes in fuel cells [13]. However, these defects have been shown to have detrimental effects on their mechanical properties [9-11].

The present study investigates the effect of carbon vacancy defects on the Young's modulus of SWCNTs through Molecular Dynamics (MD) simulations. MD simulations provide in detail the dynamic individual particle motions and structure developments as a function of time [14] and therefore serve as a great tool to study the properties of a material system. The CNT investigated in this work is the (6, 6) single wall carbon nanotube (SWCNT).

2. Creation of the Atomistic Model Configurations

2.1 Creation and Simulation of Pure SWCNT Atomistic Model Configuration

All molecular models were created in Materials Studio and molecular modeling analysis simulations were conducted using Discover module by Accelrys Inc. A unit cell of the (6, 6) SWCNT was created in Materials Studio followed by the generation of a super cell build with 10 SWCNT unit cells. The resulting carbon nanotube had a length of 23.4 Å, diameter of 8.14 Å, and a bond length of 1.42 Å. Figure 1 shows the unit cell and super cell generated and modeled with Materials Studio. The potential energy of the models is characterized by the COMPASS force field [15] with the non-bond energies characterized by the Vander Walls and Coulomb's interactions.

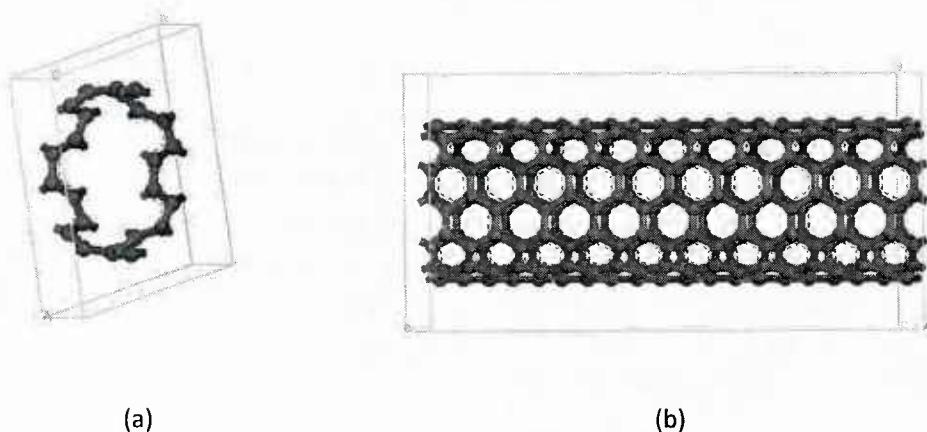


Figure 1: (a) Unit cell, (b) Supercell of the SWCNT

All angles of the cell were made equal to 90 degrees to ensure a rectangular box. The super cell molecular structure was minimized to obtain the most stable energy configuration. A cascade of the steepest descent minimization method and the Fletcher-Reeves method were used for this minimization. A typical potential energy profile during the energy minimization is shown in Figure 2.

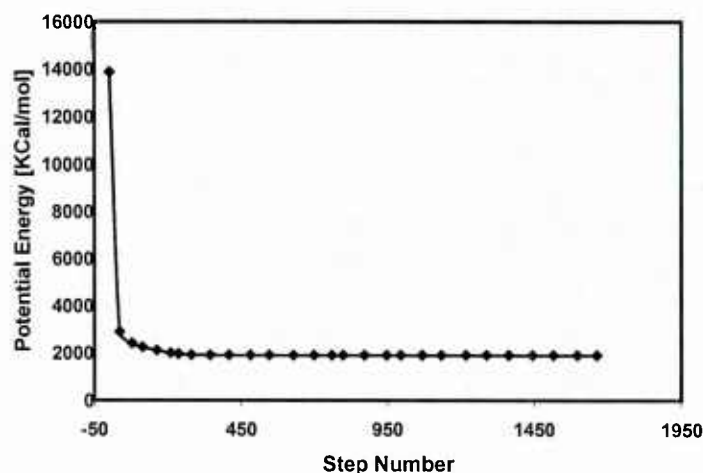


Figure 2: Potential energy profile during minimization of SWCNT

The required initial density of the cell material configuration was obtained by varying the volume of the cell. This was done by varying the lattice dimensions of the cell. Density is calculated from the total atomic material mass of the constituents in a simulation cell divided by the volume of the cell. NPT statistical ensemble which keeps the number of molecules, pressure and temperature constant but allows the volume of the cell to vary was used. The density therefore changes over the duration of the MD simulation and the reported density is the averaged density over the duration of the simulation.

As expected with a constant mass of the SWCNT defined by the super cell structure, as the volume was decreased, the density of the bounding cell increased and vice versa. However, it was observed that below certain cut-off dimensions of the bounding cell, energy minimization resulted in distortion of the nanotube structure. This made it impossible to obtain the targeted physical density of 1.9 g/cm^3 (SWCNT physical density), since this resulted in distortion of the nanotube structure. The distorted SWCNT structures obtained after the full minimization convergence are not however representative of the physical configuration of the SWCNT. Figure 3 shows examples of distorted SWCNT nanotube structures obtained after full energy minimization convergence.

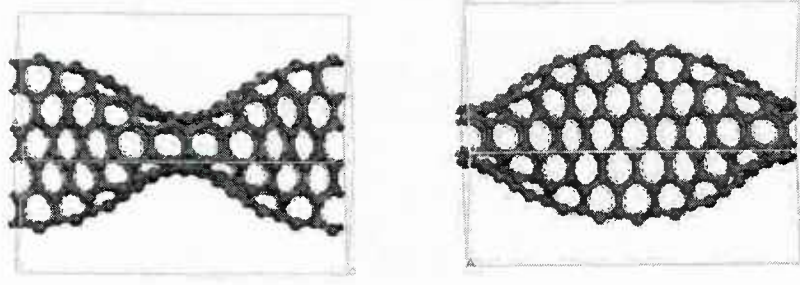


Figure 3: Distorted SWCNT after minimization

In order to avoid these unphysical distortions seen during the energy minimization, mechanical property predictions for the Young's modulus was determined at several densities that had undistorted minimized structures. The predicted Young's Modulus values from these different densities were then extrapolated to obtain the Young's Modulus at the physical density of 1.9 g/cm^3 . MD simulation analyses were conducted at a temperature of 298 K and a pressure of 1 atmosphere for 50 ps (50,000 fs) with a time step of 1 fs. All simulation analyses employed NPT ensemble with the Anderson temperature control method [16, 17] and the Berendsen pressure control method [15]. Trajectories were saved at every 5,000 steps resulting in 10 frames over the entire time duration of the simulation. These 10 molecular trajectory frames were used in the computational analysis for the predicted Young's modulus. Young's modulus as predicted and obtained from Accelrys analysis computations was calculated from the Lamé constants λ and μ as given by Equation 1.

$$E = \mu \left(\frac{3\lambda + 2\mu}{\lambda + \mu} \right) \quad (1)$$

A 6×6 matrix of elastic constants generated by the Accelrys analysis computations was analyzed to obtain the modulus in various directional orientations.

Table 1 lists the dimensions of the nanotubes used with "a", "b" and "c" being the dimensional length of the bounding cell in the "x", "y" and "z" axis respectively.

The lattice parameters for the configuration SWCNT1 represent the threshold below which further minimization of the cell resulted in an unphysical distortion of the nanotube as discussed earlier.

Table 1: Lattice parameters of SWCNT cell structures used to investigate the Young's Modulus

Lattice parameters[Å]	SWCNT1	SWCNT2	SWCNT3	SWCNT4
a	11.483	12.5	13.5	14.5
b	11.483	12.5	13.5	14.5
c	25.5951	25.5951	25.5951	25.5951

2.2 Creation and Simulation of Defective SWCNT Atomistic Models

The defective single walled carbon nanotubes (DSWCNT) with vacancy defects were created in the same way as the pure SWCNT. The vacancy defects in the SWCNT structure were introduced by removing some carbon atoms in the "z" dimensional axis of the tube. As mentioned earlier in section 1, these vacancies of carbon atoms are sometimes advantageous from the chemical point of view, because the affinity of the structure is increased at these vacant sites. However, mechanically these defects make the structure weaker and results in a decrease of the mechanical properties.

Two types of investigations were conducted on the effect of carbon vacancy defects on the predicted Young's Modulus of SWCNT. The first investigation is to determine the effect of the position of the carbon vacancy defect on the Young's modulus of the nanotube. This was modeled by moving the vacancy defect around the lattice length of the nanotube in the "z" dimension. The second investigation is to find the effect of the number of vacancy defects on the Young's modulus. The number of defects incorporated into the tube was altered. Young's modulus was therefore investigated for different number of defects in the tube. As with the pure SWCNT, the Young's modulus was determined at different densities and extrapolated to the required physical SWCNT density of 1.9 g/cm^3 to avoid the unphysical distortions during energy minimization as discussed earlier. The lattice parameters used in the SWCNT model configurations with carbon vacancy defects were the same as in the pure case and are as listed in Table 1. A similar procedure as discussed earlier was employed for the creation of the molecular model with the carbon vacancy defects. A unit cell was created followed with a super cell with 10 SWCNT units generated with all angles equal to 90 degrees. Vacancy defects in the SWCNT structure were formed by removing some carbon atoms from the rings. The structure with carbon vacancy defects was then minimized. The energy minimization was monitored to avoid cell dimension configurations that would lead to unphysical distortion of the SWCNT structures. MD simulation analyses were conducted at a temperature of 298 K for 50 ps (50,000 fs) with a time-step of 1 fs. 10 trajectories of the molecular configurations during the dynamic analysis were

saved and used for the analysis run to estimate the predicted Young's modulus of SWCNT with various vacancy defects.

3 Results and Discussions

The predicted Young's modulus from various MD simulations with the carbon vacancy defects are compared with those for a pure, non-defective SWCNT. The findings and inferences from the several simulation analyses are discussed next.

3.1 Position of the Defect

The carbon vacancy defect was moved along the “z” axis dimension of the SWCNT to investigate the effect of the position of the vacancy defect within the SWCNT molecular structure on the Young's modulus. Due to the symmetrical nature of the SWCNT, a carbon vacancy defect in the second ring would be equivalent to the carbon vacancy defect in the eighth ring. In the same regard, defects in the third and seventh ring are equivalent, so are defects in the fourth and sixth. For this reason, the defects were moved only from the second ring to the fifth ring. Any further than this will just be duplication for the SWCNT length considered in the present study. The defect used in this work consisted of removing adjacent vertical carbon atoms in a particular ring as shown in Figure 4.

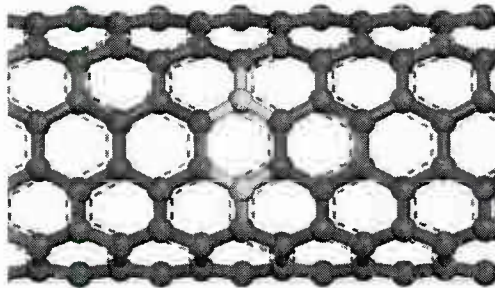


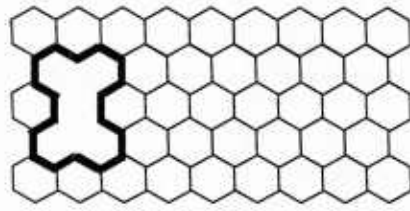
Figure 4: SWCNT with two carbon atoms removed

Removal of one carbon atom from an enclosed ring results in the breaking of three C-C bonds. The carbon atoms removed and the bonds broken are highlighted in Figure 4. Removal of two carbon atoms therefore results in the breaking of six C-C bonds. Carbon atoms were not removed from the first or the ninth rings because that would have resulted in the breaking of fewer C-C bonds since the first and ninth rings were the boundary rings, and will therefore not augur well for accurate comparisons. It must be noted however that the results are not expected to be the same if the carbon atoms were removed horizontally since this would result in a different shape of the defect. The movement of the carbon vacancy defect along the “z” dimension of the tube is schematically shown in Figure 5.

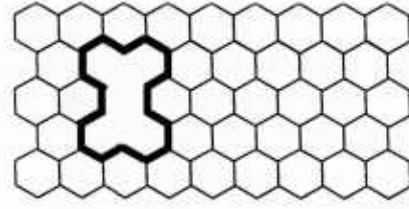
Table 2 shows the Young's modulus at different positions on the SWCNT for adjacent vertical carbon atoms removed and Table 3 shows the Young's modulus at different positions of the SWCNT for adjacent horizontal atoms removed.

Table 2: Effect of the position of the defect on Young's Modulus (adjacent vertical atoms removed)

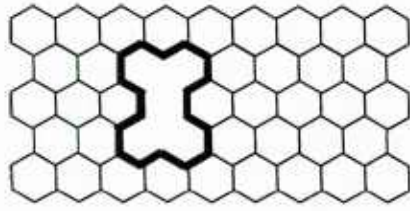
Position of defect	Young's Modulus [GPa]
Second ring (Figure 5a)	704.4
Third ring (Figure 5b)	704.2
Fourth ring (Figure 5c)	704.2
Fifth ring (Figure 5d)	704.1



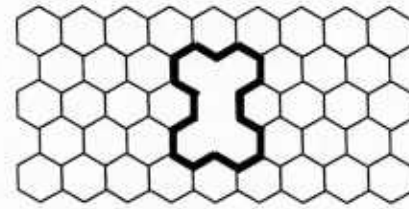
(a)



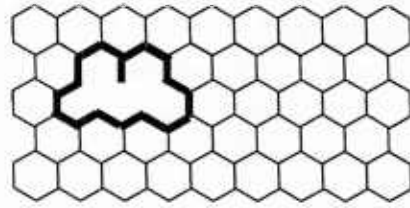
(b)



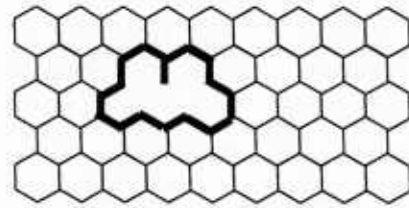
(c)



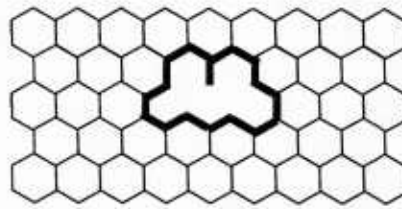
(d)



(e)



(f)



(g)

Figure 5: Various positions of adjacent defects in the SWCNT: (a) Vertical defect in second ring, (b) Vertical defect in third ring, (c) Vertical defect in fourth ring, (d) Vertical defect in fifth ring, (e) Horizontal defect in the second and third rings, (f) Horizontal defects in the third and fourth rings, (g) Horizontal defects in the fourth and fifth rings

Table 3: Effect of the position of the defect on Young's Modulus (adjacent horizontal atoms removed)

Position of defect	Young's Modulus [GPa]
Second and third rings (Figure 5e)	735.4
Third and fourth rings (Figure 5f)	735.6
Fourth and fifth rings (Figure 5g)	735.6

From Table 2 and 3 it is clear that for a defect with the same shape, its position in the SWCNT had no effect on the Young's modulus. This can be attributed to the fact that, for the same shape and size of vacancy defect, regardless of its position, the same type of bonds are broken and therefore the weaknesses introduced are equivalent. The results were however different when the shape of the defect was changed and the number of defects kept the same. This is evident from the fact that the average Young's modulus for the SWCNT with two adjacent vertical defects (Table 2) is 704.2 GPa while the average Young's modulus for the SWCNT with two adjacent horizontal defects (Table 3) is 735.5 GPa. The results from the simulations indicate that moving the defect along the "z" dimension of the tube therefore has no effect on the Young's modulus for the defects of the same shape, and the removed carbon atoms are enclosed.

3.2 Number of Defects

The effect of the number of defects on the Young's modulus was also studied. Simulation runs were conducted for nanotubes with no defects, two defects, four defects, six defects and eight defects. Fig 6 shows the configuration of these various defects in the molecular structure of SWCNT. All these were taken to be adjacent vertical defects.

As mentioned in section 2.1, Young's modulus was determined at densities for which the lattice parameters did not result in distortion of the SWCNT during minimization. These were then extrapolated to the actual nanotube density of 1.9 g/cm^3 to obtain the Young's modulus at this physical density. In obtaining the relationship between the density and Young's modulus of the nanotube, the point (0, 0) was added to the simulated data points in figures 7 thru 11. This is due to the fact that without a SWCNT, the density of the cell is zero. Figures 7 through 11 show the simulated densities and the corresponding Young's modulus obtained for the cases of zero defect, two defects, four defects, six defects and eight defects respectively. The extrapolated Young's modulus for the density of 1.9 g/cm^3 is also shown in all cases and the corresponding values are presented in Table 4.

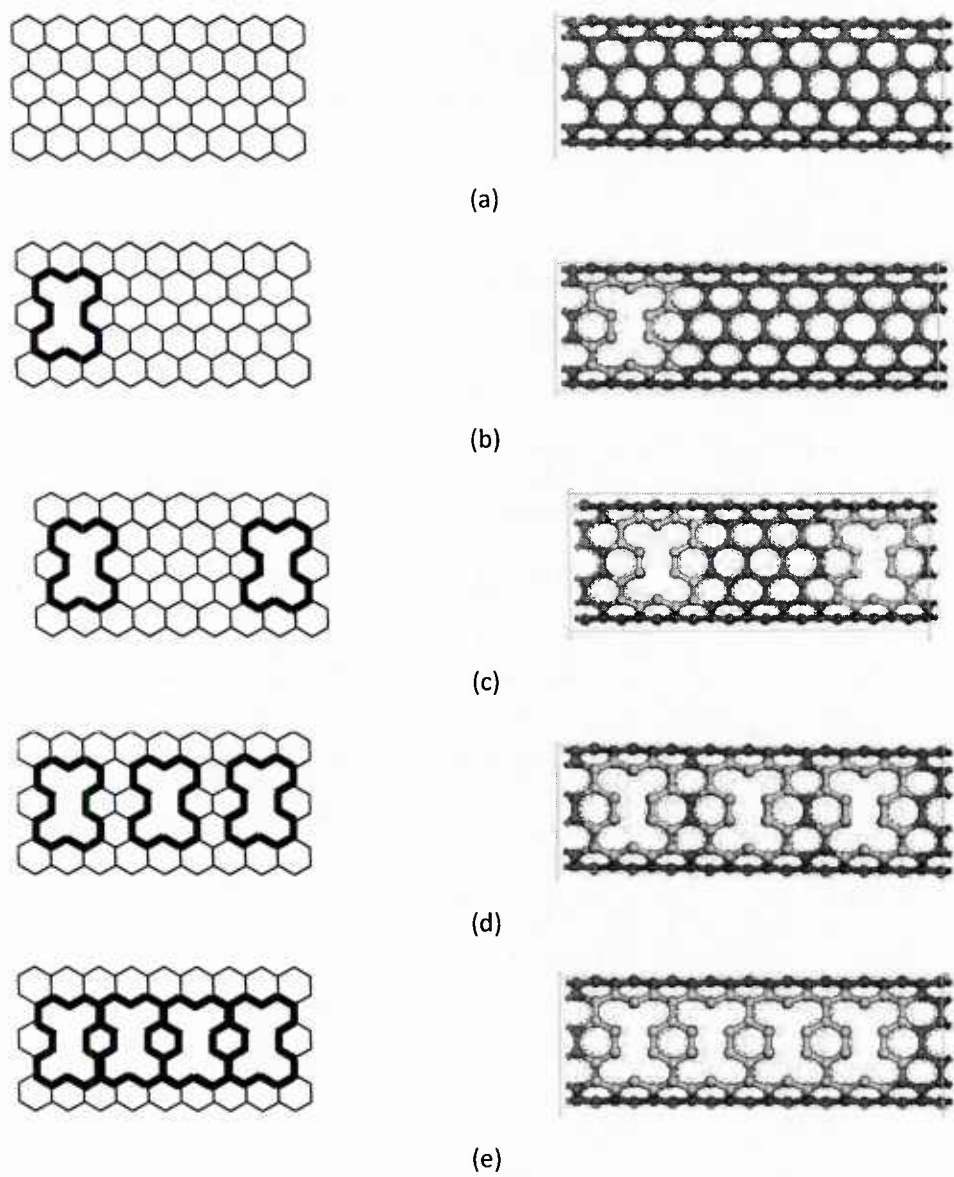


Figure 6: Various numbers of defects in the SWCNT: (a) No defect, (b) 2 defects, (c) 4 defects, (d) 6 defects and (e) 8 defects

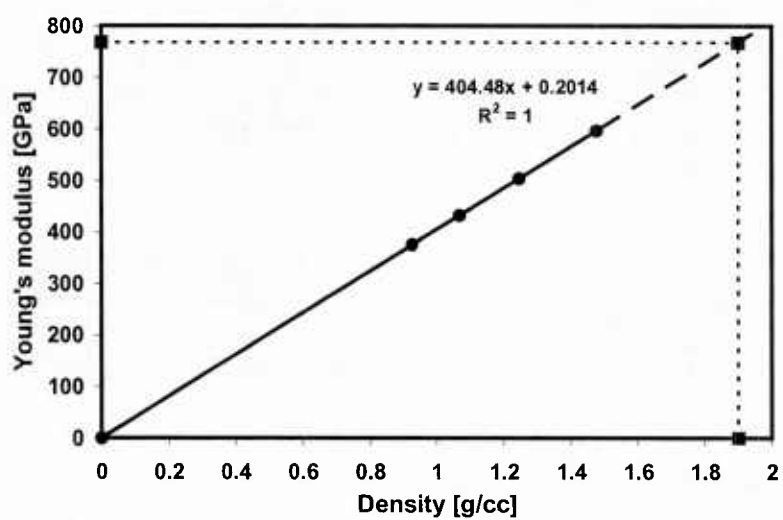


Figure 7: Young's modulus of SWCNT with no defect (pure nanotube)

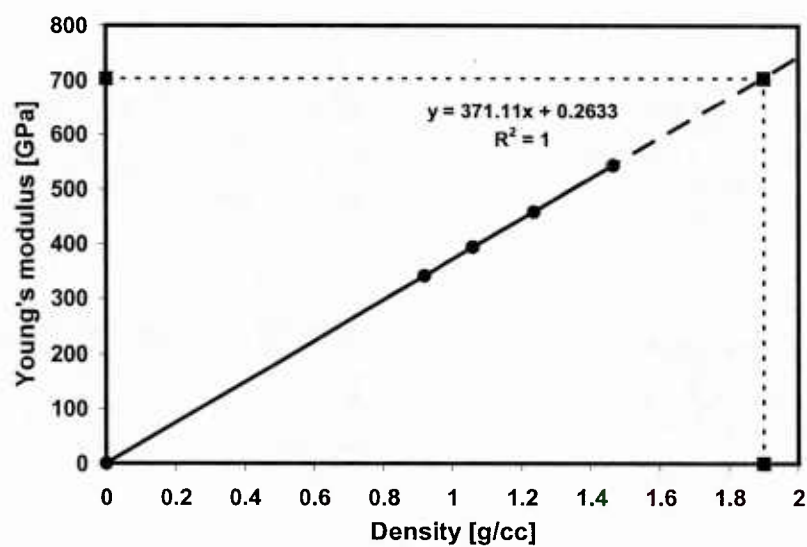


Figure 8: Young's modulus of SWCNT with 2 carbon vacancy defects

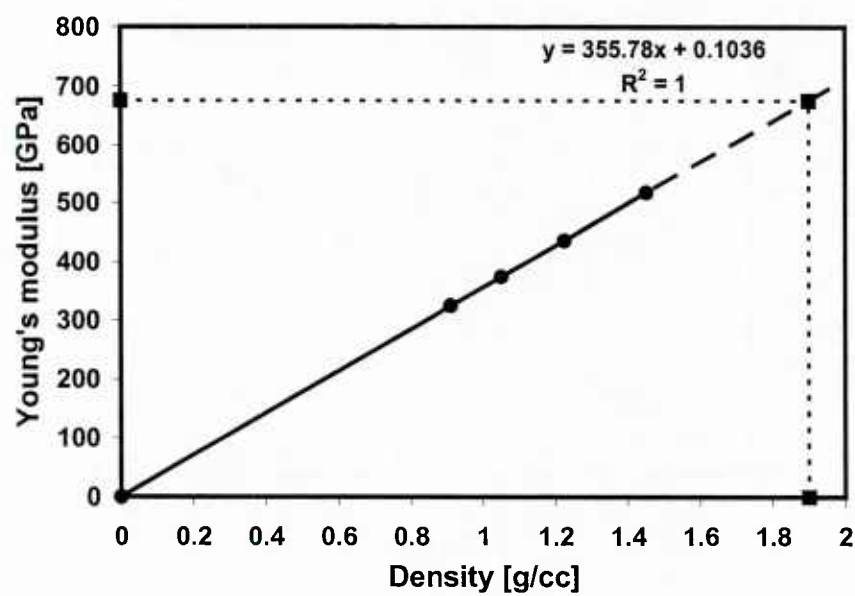


Figure 9: Young's modulus of SWCNT with 4 carbon vacancy defects

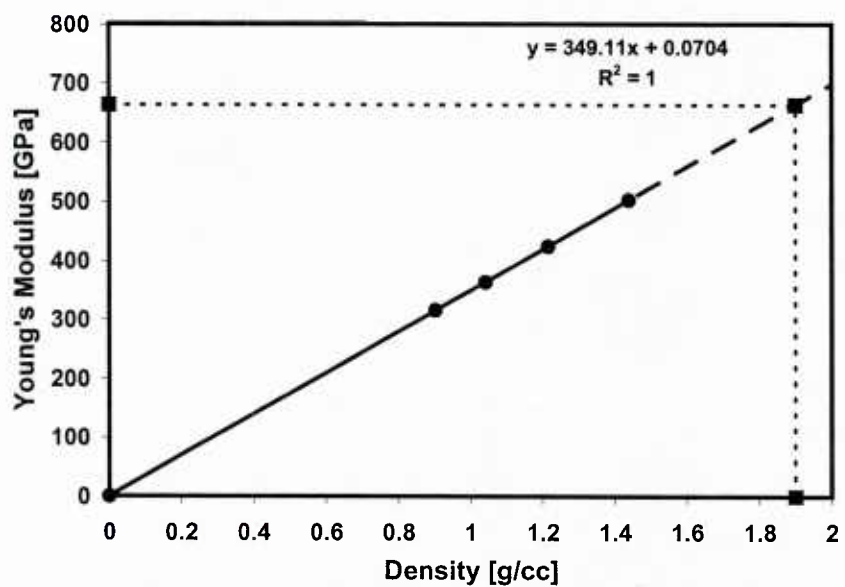


Figure 10: Young's modulus of SWCNT with 6 carbon vacancy defects

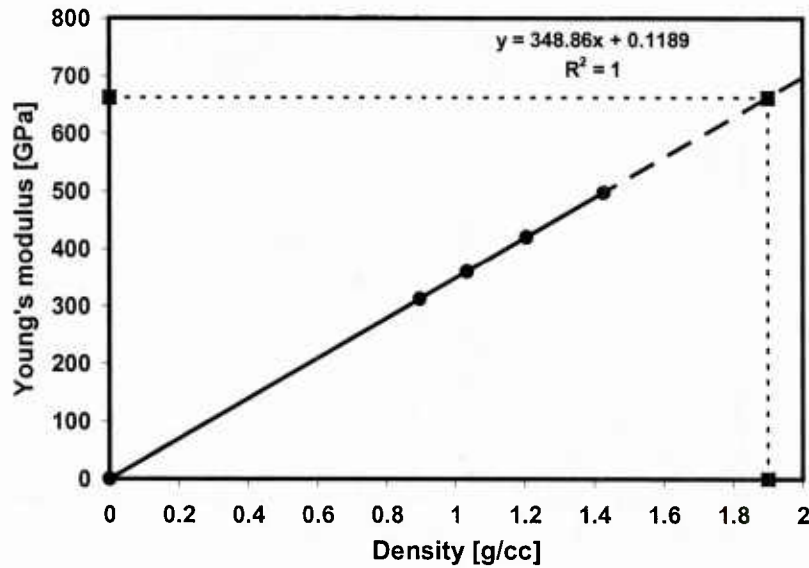


Figure 11: Young's modulus of SWCNT with 8 carbon vacancy defects

Table 4: Effect of number of defects on the extrapolated Young's modulus at the physical density of 1.9 g/cm³

Number of defects	Young's Modulus [GPa]
0	768.7
2	705.4
4	676.1
6	663.4
8	662.9

The effect of the number of carbon vacancy defects in a SWCNT on the Young's modulus of the SWCNT is shown in Figure 12.

It is clear from figure 12 that the Young's modulus of the defective nanotube (SWCNT) decreases with increasing number of defects. This result conforms well to the reduction in fracture toughness for larger defect radius reported in the literature [11]. This phenomenon further supports the fact that defects in the SWCNT could be a potential cause of the disparity in results obtained from experiments and molecular modeling simulations of polymer

nanocomposites containing CNTs. A noticeable observation from Figure 12 though is that, after a certain number of defects (six in this case); there is no more noticeable reduction in the Young's modulus. The variation of the defects percentage in the SWCNT and the corresponding percentage reduction in Young's modulus of the SWCNT is shown in Table 5.

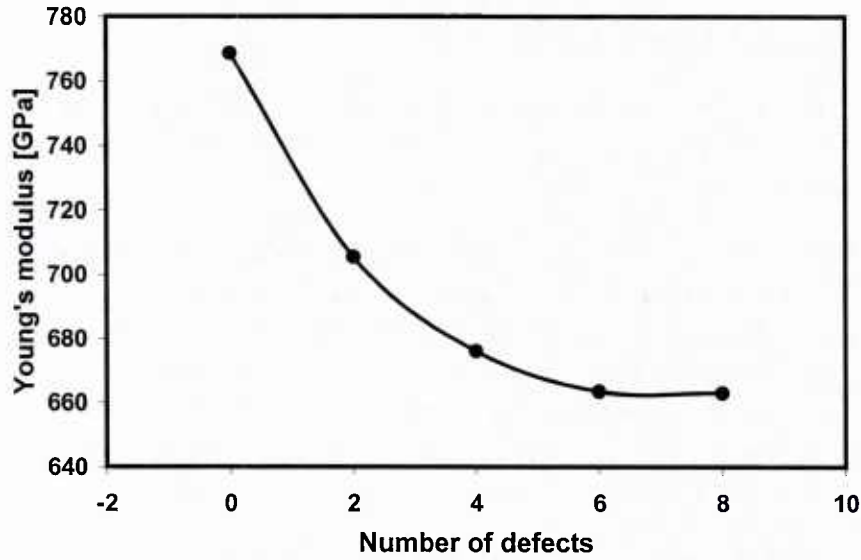


Figure 12: Effect of number of defects in SWCNT on Young's modulus

Table 5: Variation of SWCNT percentage defects and Young's Modulus

% defects in SWCNT	% reduction in Young's Modulus [GPa]
0.83	8.24
1.66	12.05
2.5	13.7

4. Summary

Experimental results obtained for the mechanical properties of SWCNTs and polymer nanocomposites containing SWCNTs have tended to be lower than the results obtained from MD simulations. One potential cause for this disparity is the presence of molecular defects in the SWCNTs.

In this work, the effect of carbon vacancy defects on the Young's modulus of a 10-unit (6, 6) SWCNT was investigated using MD simulations. The defect in this work was made up of carbon vacancies. This was achieved by removing adjacent vertical carbon atoms from the SWCNT molecular ring. Two types of investigations were performed:

1. The effect of the position of the carbon vacancy defect, and
2. The effect of the number of the vacancy defects.

The SWCNT models were created and minimized. MD simulations were run for 50 ps with a time step of 1 fs. Molecular trajectories were saved every 5000 steps and used for the evaluation of the Young's modulus.

To study the effect of the position of the defect, the defect was moved along the "z" dimension of the tube. The results indicated that the position of the defect did not cause any change in the Young's modulus. The results varied by less than 1% (701-704 GPa) for the case where adjacent vertical atoms were removed. However, the results were different when the shape of the defect was changed by removing adjacent horizontal atoms (735.4-735.6 GPa). It must also be noted that these defects are in enclosed rings, not on the boundary rings. This ensures that equal numbers of C-C bonds are broken in all cases.

The effect of the number of carbon vacancy defects was investigated by varying the number of defects incorporated into the SWCNT. Young's modulus was obtained for nanotubes with no defect, two defects, four defects, six defects and eight defects. The analysis results clearly indicate that as the number of defects increased, the Young's modulus decreased. Incorporation of six defects (2.5% defects) reduced the SWCNT nanotube modulus by about 13% (from 767 GPa to 663 GPa). Furthermore, the analysis results indicate that increasing the number of defects beyond six not to result in any further reduction of the Young's modulus.

References

- [1] Iijima S., *Helical Microtubules of Graphitic Carbon*, Letters to Nature, Volume 354 (56), (1991) 56-58.
- [2] Zhou Y. X., Wu P. X., Cheng Z. Y., Ingram J., Jeelani S., *Improvement in electrical, thermal and mechanical properties of epoxy by filling carbon nanotube*, Express Polymer Letters, Volume 2, Number 1, (2008) 40-48.
- [3] Demczyk B. G, Wang Y. M, Cumings J, Hetman M, Han W, Zettl A, Ritchie R, O. *Direct mechanical measurement of the tensile strength and elastic modulus of multiwalled carbon nanotubes*. Materials Science and Engineering A, Volume 334, (2002) 173-178.

- [4] Ning H., Zen M., Cheng Y., Go Y., Hisao F., Toshiyuki H., *The electrical properties of polymer nanocomposites with carbon nanotube fillers*, Nanotechnology, Volume 19, (2008) 215701.
- [5] Yue H., Elliot J., *Molecular dynamics simulations of the elastic properties of polymer/carbon nanotube composites*, Computational Materials Science, Volume 39, (2007) 315-323.
- [6] Zhu R., Pan E., Roy A. K., *Molecular dynamics study of the stress-strain behavior of carbon-nanotube reinforced EPON 862 composites*, Materials Science and Engineering A, Volume 447, (2007) 51-57.
- [7] Gou J., Minaie B., Wang B., Liang Z., Zhang C., *Computational and experimental study of interfacial bonding of single-walled nanotube reinforced composites*, Computational Materials Science, Volume 31, (2004) 225-236.
- [8] Komuves F., *Prediction of mechanical properties of EPON 862 (DGEBA) cross-linked with curing agent W (DETA) and (6, 6) SWCNT using MD simulations*, PhD Dissertation, Mechanical Engineering 2009.
- [9] Li Z., Wang C. Y., Ke S. H., Yang W., *First principles study for transport properties of defective carbon nanotubes with oxygen adsorption*, The European Physical Journal B, Volume 69, (2009) 375-382.
- [10] Mielke S. L., Troya D., Zhang S., Li J. L., Xiao S., Car R., Ruoff R. S., Schatz G. C., Belytschko T., *The role of vacancy defects and holes in the fracture of carbon nanotube*. Chemical Physics Letters, Volume 390, (2004) 413-420.
- [11] Mielke L. S., Zhang S., Khare R., Troya D., Ruoff R. S., Schatz G. C., Belytschko T., *Mechanics of defects in carbon nanotubes: atomistic and multiscale simulations*, Physical Review B, Volume 71, (2005) 115403.
- [12] Chico L., Crespi V. L., Benedict L. X., Louie S. G., Cohen M. L., *Pure carbon nanoscale devices: nanotube heterojunctions*, Physical Review Letters, Volume 76, Number 6, (2006) 971-974.
- [13] Park Y., Lahaye R. J., Lee Y., *Adsorption of Pt on defective carbon nanotube walls: a DFT approach*. Computer Physics Communications, Volume 177, Issues 1-2, (2007) 46.
- [14] Liu W.K., Karpov E. G., Zhang S., Park H. S., *An introduction to computational nanomechanics and materials*, Computer Methods in Applied Mechanics and Engineering, Volume 193, (2004) 1529-1578.

- [15] Sun H., *The COMPASS force field: parameterization and validation for phosphazenes*, Computational and Theoretical Polymer Science, Volume 8, Number 1/2, (1998) 229-246.
- [16] Hu Y., Sinnott B. S., *Constant temperature molecular dynamics simulations of energetic particle-solid collisions: comparison of temperature control methods* Journal of Computational Physics, Volume 200, Issue 2, (2004) 251-266.
- [17] Andersen C. H., *Molecular dynamics simulations at constant pressure and/or temperature*. Journal of Chemical Physics, Volume 2, Number 4, (1980) 2384-2393.

Predictive Mechanical Properties of EPON 862 (DGEBF) cross-linked with Curing Agent W (DETDA) and SWCNT using MD Simulations – Effect of Carbon Vacancy Defects

Authors: R. Mohan, E. Fefey, A. Kelkar, North Carolina A&T State University

Published Referred AIAA Proceedings: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, April 2012.

ABSTRACT

Molecular Dynamics (MD) simulations are a viable alternative to experimental methods to obtain mechanical properties of EPON 862-DETDA-SWCNT composites. This paper investigates the effect of SWCNT carbon vacancy defects on the Young's modulus of the EPON 862-DETDA-SWCNT composite using MD simulations performed via Accelrys. For a composite with 7-12 weight% of SWCNT, 2 carbon vacancy defects in the SWCNT is found to reduce the Young's modulus by 13-18%, while 4 carbon vacancy defect in the SWCNT reduced the Young's modulus of the composite by 21-30%. This clearly indicates that carbon vacancy defects are one potential cause for the disparity, and lower Young's modulus values of Epoxy-SWCNT composites cited in the literature.

Introduction

The inclusion of nanomaterial constituents into polymeric resin systems has gained significant attention because of the enhancement in mechanical and thermophysical properties that are attained [1- 3]. Carbon nanotubes (CNTs), in particular have been reported to increase the mechanical properties of its parent polymers significantly [4, 5]. The mechanical properties of CNT-reinforced epoxy systems such as the EPON 862-DETDA-SWCNT composite system have been studied through theoretical, experimental and computational analysis [6-9]. Experimental mechanical properties obtained and cited in the literature are generally lower than the values obtained from computational analysis [10-12]. Some reasons for this disparity have been attributed to the fact that experimental macroscopic coupons have CNTs dispersed in various orientations, while the computational molecular models usually have the CNTs uni-directionally aligned.

Computational molecular models are also normally small built with few cured epoxy molecular structures. The high computational costs restrict the use of larger molecular cure network models that could better represent the fully cured epoxy structure. Another reason for this disparity is the fact that experimental processes could possibly introduce defects into the CNTs while

computational models are based on ideal CNT molecular configuration. As cited in the literature, carbon vacancy defects in the CNT molecular structure have been known to cause changes in the properties of CNTs [13-16]. A study of the effect of carbon vacancy defects on the Young's modulus of the CNT has been presented in an earlier work by the present authors [17]. Chemically, these defects have been reported to enhance the affinity of the CNTs at the defective site making them most suitable as Platinum carrier electrodes in fuel cells [18]. However, these defects have been shown to have detrimental effects on the mechanical properties as cited in the literature [10-12].

The present paper investigated the effect of carbon vacancy defects in the ideal CNT molecular configuration on the fundamental mechanical properties of the CNT-epoxy nanocomposite system through molecular dynamics simulations. The defects were in the form of loss of carbon atoms in the ideal CNT structure creating carbon vacancy defects. The polymer nanocomposite system used in the present work is the (6, 6) single walled carbon nanotube (SWCNT) and the matrix system was diglycidylether of bisphenol F (EPON 862) cross linked with diethyltoluenediamine (DETDA).

Methodology

All molecular models in this work were created in Materials Studio and molecular modeling analysis simulations were conducted using Discover and Amorphous module from Accelrys Inc. Materials Studio is a graphical user interface that allows construction of atomistic models and set up the analysis required for characterization of these molecular models and prediction of mechanical properties. MD simulations provide in detail the individual particle motions and structure developments as a function of time [19], and therefore serve as a great tool to study the properties of a material system at the molecular/atomistic level.

Molecular Model of EPON 862-DETDA-SWCNT Composite

The recommended weight ratio of EPON 862 to DETDA for a fully cured composite during processing is 100:26.4 [10, 11]. Figure 1 shows the molecular structures of EPON 862 and DETDA from Materials Studio.

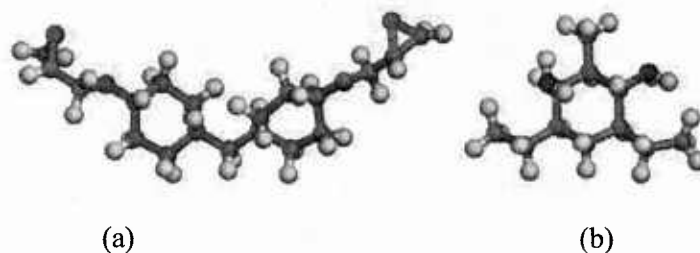


Figure 1. (a) EPON 862 and (b) DETDA molecular structures from Materials Studio

The molecular weight of EPON 862 is 312, and DETDA has a molecular weight of 178. Based on this, the molecular ratio of the fully cured composite was formulated to be 2 molecules of EPON 862 linked with 1 molecule of DETDA, giving nearly the same recommended weight ratio of 100:26.4 that is employed in actual processing for the cured epoxy molecular structure. The fully cured composite molecular model was therefore constructed with 8 molecules of EPON 862 and 4 molecules of DETDA.

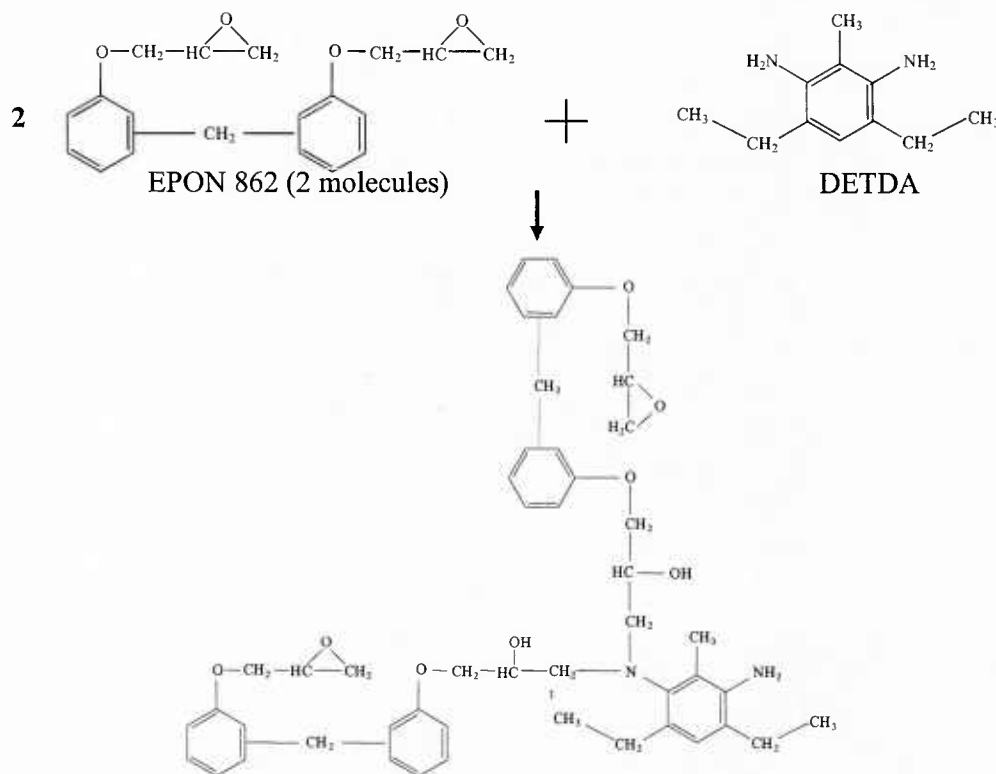


Figure 2. Cross-linking of 2 EPON-862 molecules with 1 DETDA molecule

The DETDA molecule has 2 amine (NH_2) groups and the EPON-862 molecule has 2 epoxide (CHCH_2O) groups. Each of these amine groups in the DETDA can react with 2 epoxide groups of the EPON-862 [9].

The cross linking process was initiated by bonding 2 epoxide groups from 2 different EPON-862 molecules with 1 of the amine groups. This formed a crosslink of the 2 EPON-862 molecules at the resulting N-atom of the DETDA. This reaction is shown in Figure 2. This cross linking process was repeated with different EPON-862 molecules for all the 4 DETDA molecules, so each DETDA molecule cross linked 2 EPON-862 molecules.

At this stage, each of the 8 EPON-862 cross linked molecules had one un-bonded epoxy group while each of the 4 DETDA molecules also had an un-bonded amine group. 4 of these epoxy groups were bonded with the 4 free amine groups resulting in a ring of 8 molecules of EPON-862 and 4 molecules of DETDA. Figure 3(a) shows a diagrammatic representation of the ring while Figure 3(b) shows the actual ring from Materials Studio. A unit cell of the SWCNT is also shown in Figure 3(c).

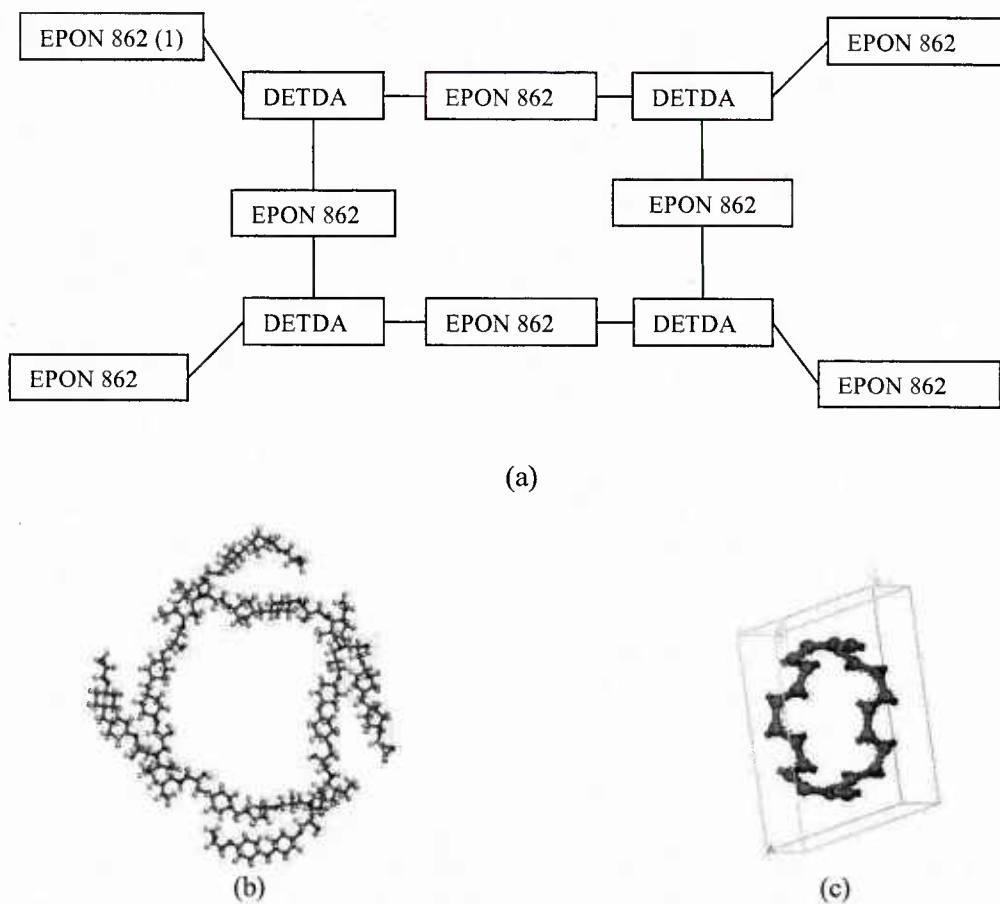


Figure 3. (a) Diagrammatic representation of fully cured (8:4) EPON 862 cross linked with DETDA, (b) Fully cured (8:4) EPON 862 cross linked with DETDA from Materials studio, (c) SWCNT unit cell

Three polymer nanocomposite molecular models with SWCNT weight percentages between 7% and 12% were employed in the present investigations. The three molecular models of the

defective SWCNT (DSWCNT) and cured epoxy composite system used in the present study had the following configurations and SWCNT weight percentages based on the cell dimensions of the molecular models:

1. Molecular Model Configuration A: 3 unit cells of DSWCNT and 2 fully cured epoxy matrix corresponding to the CNT weight percentage of 11.28-11.58%
2. Molecular Model Configuration B: 4 unit cells of DSWCNT and 3 fully cured epoxy matrix corresponding to the CNT weight percentage of 10.34-10.49%
3. Molecular Model Configuration C: 4 unit cells of DSWCNT and 4 fully cured epoxy matrix corresponding to the CNT weight percentage of 7.95-8.08%

The potential energy of the molecular models was characterized by the COMPASS force field [20], with the non-bond energies characterized by the Vander Walls and Coulomb's interactions. The dynamic analysis was performed using the NPT ensemble in conjunction with the Anderson temperature control method [21, 22] and the Berendsen pressure control method [20].

The different molecular model configurations were minimized to obtain the lowest energy configuration. A cascade of the steepest descent minimization method and the Fletcher-Reeves method were used for the minimization. The minimized energy molecular models were subsequently equilibrated with the NVT ensemble for 100 ps at 298 °K. A sample simulation molecular cell is shown in Figure 4.

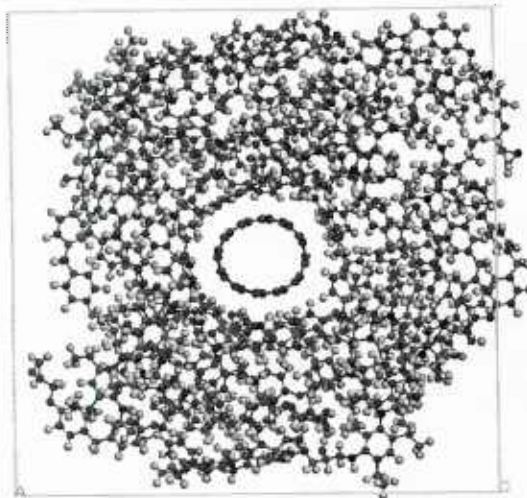


Figure 4. Simulation cell showing the CNT embedded in the fully cured epoxy matrix

Simulated annealing was used to mimic the curing cycle of EPON 862-DETDA-SWCNT composite and to ensure that the final configuration had the lowest energy possible. A characteristic of simulated annealing is lowering the temperature slowly in stages to allow thermal equilibrium of the molecular configuration to be attained at each stage. At high temperatures, molecules move freely, but as the temperature decreases, thermal mobility decreases, and the molecules tend to align in a state of minimum energy as long as the temperature is decreased slowly [23]. The molecular cell was heated to 498 °K, and the temperature was dropped to 298 °K in steps of 10 °K using the NPT ensemble with a specified pressure of 0.0001 GPa (1 atm). MD simulation analysis was conducted at each temperature during the simulated annealing process for 200 ps (200,000 fs) with a time step of 1 fs. The final molecular structure of each simulated annealing temperature step was used as the starting structure of the next step. The molecular configuration density reported at each simulated annealing temperature step was noted. All simulations employed NPT ensemble which keeps the number of molecules, pressure and temperature constant, but allows the volume of the cell to vary. The density therefore changes over the dynamic duration of the MD simulation and the reported density is the averaged density over the dynamic duration of the simulation. At 298 °K, an analysis of the elastic properties was performed by saving 10 trajectories and using them for the predicted Young's Modulus. Young's modulus as predicted and obtained from Accelrys analysis computations was calculated from the Lamé constants λ and μ using Equation 1.

$$E = \mu \left(\frac{3\lambda + 2\mu}{\lambda + \mu} \right) \quad (1)$$

A 6 × 6 matrix of elastic constants generated via the Accelrys analysis computations was analyzed to obtain the modulus in various directional orientations.

Defect Types and Defective SWCNT Composite Systems

The effect of two and four carbon vacancy defects in the SWCNT on the mechanical properties of the EPON-SWCNT composite was studied. The vacancy defects in the SWCNT structure was introduced by removing two adjacent vertical carbon atoms on one side of the nanotube. Because of the short length of the nanotube in the present study, the four carbon vacancy defect was obtained by the removal of two adjacent vertical carbon atoms on opposite sides of the nanotube.

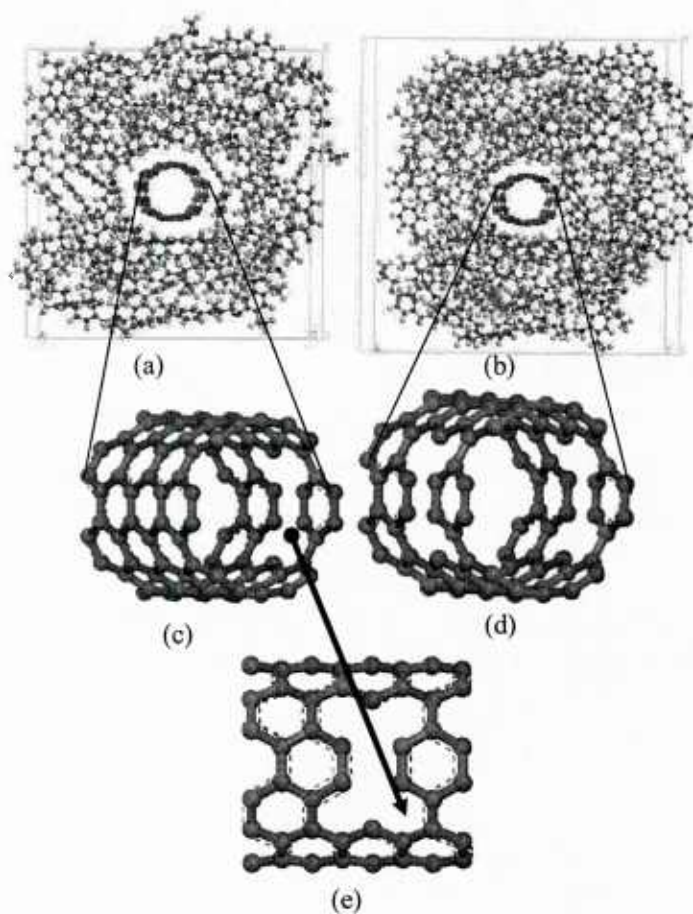


Figure 5. (a) Simulation cell with CNT having 2 defects (b) Simulation cell with CNT having 4 defects (c) Zoomed CNT showing the 2 defects (d) Zoomed CNT showing the 4 defects (e) Side view of the 2 defects

Figures 5(a) and 5(b) show the DSWCNT embedded within the epoxy molecular structure. The composite molecular configuration with the defective SWCNT and EPON configuration as previously discussed was constructed for each of the defective structures. Figures 5(a) and 5(b) presents the composite molecular structure with defective SWCNT and one of the fully cured EPON configuration employed in the present work. Figure 5(c) shows a zoomed image of the DSWCNT depicting two missing carbon atoms, while Figure 5(d) shows the DSWCNT depicting four missing carbon atoms. Figure 5(e) shows a side view of the zoomed CNT showing the two defects.

Rule of mixtures was used to obtain the density of the composite molecular configuration. The rule of mixtures density is given by Equation 2.

$$\rho_{mixture} = \rho_{SWCNT} f_{SWCNT} + \rho_{resin} f_{resin} \quad (2)$$

where $\rho_{mixture}$, ρ_{SWCNT} and ρ_{resin} are the densities of the mixture, SWCNT and the epoxy resin, respectively, and f_{SWCNT} and f_{resin} are the volume fractions of the SWCNT and the epoxy resin, respectively.

With the SWCNT having a density of 1.9 g/cm³ and the epoxy resin having a density of 1.2 g/cm³, the rule of mixtures densities of the composite configurations were calculated using Equation 2.

The composite rule of mixture density for different molecular configurations and weight percentages of SWCNT is presented in Table 1.

Table 1. Rule of mixtures densities of the composites

Model	Weight percentage of SWCNT	Rule of mixtures density of composite [g/cm ³]
No defect		
3 units SWCNT, 2 units resin	11.87	1.2549
4 units SWCNT, 3 units resin	10.69	1.2492
4 units SWCNT, 4 units resin	8.24	1.2376
2 defects		
3 units DSWCNT, 2 units resin	11.58	1.2539
4 units DSWCNT, 3 units resin	10.49	1.2482
4 units DSWCNT, 4 units resin	8.08	1.2368
4 defects		
3 units DSWCNT, 2 units resin	11.28	1.2521
4 units DSWCNT, 3 units resin	10.34	1.2475
4 units DSWCNT, 4 units resin	7.95	1.2359

Model Configuration A: 3 units of SWCNT, 2 units of resin

Model Configuration B: 4 units of SWCNT, 3 units of resin

Model Configuration C: 4 units of SWCNT, 4 units of resin

The Young's Modulus at the rule of mixtures density was obtained by conducting simulated annealing analysis as discussed earlier at three different lattice configurations for each composite molecular model configuration (corresponding to different weight percentage of CNT) and extrapolating to the corresponding rule of mixtures density.

Results

The Young's modulus at the rule of mixture density for the three composite molecular configurations with different weight% of defective SWCNT studied are shown in Table 2 and Figure 6. Also shown are the Young's modulus obtained for the composite system with pure, non-defective SWCNT [10].

Table 2. Evaluated Young's modulus at Rule of Mixture Density for the composites studied

CNT weight %	Young's Modulus [GPa]		
(No defect)			
	No defect [10]	2 defects	4 defects
11.87	74	60.75±1.96	52.07±1.82
10.69	65	54.77±1.12	48.68±1.06
8.24	52.5	45.38±1.05	41.00±1.69

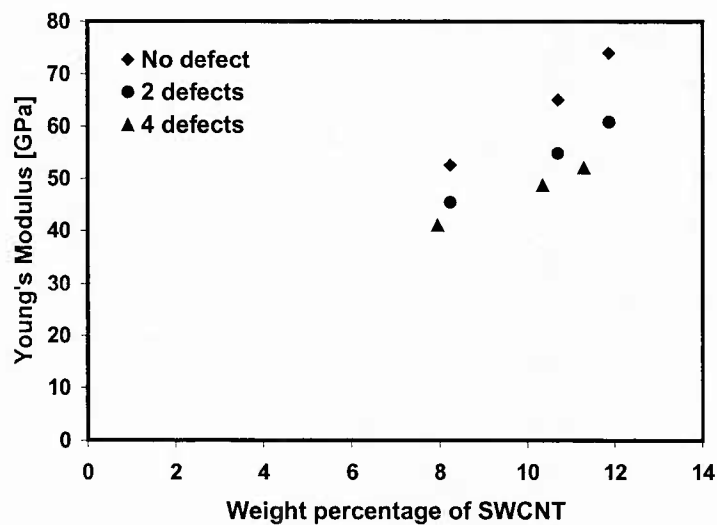


Figure 6. Variation of Young's modulus with SWCNT weight percent for No defect, 2 defects and 4 defects

The following can be inferred from Table 2 and Figure 6.

- For a given initial weight percentage of SWCNT, the Young's modulus of the EPON-SWCNT composites with non-defective SWCNT were the highest (52.5-74 GPa, for the three weight % of the SWCNT studied in the three molecular model configurations).
- The predicted Young's Modulus for the composites with 2 SWCNT vacancy defects was lower (45.38-60.75 GPa, for the three weight% of SWCNT studied).
- The predicted Young's Modulus was the lowest for the composite system composed of EPON and SWCNT with 4 carbon vacancy defects (41.00-52.07 GPa, for the three weight % of SWCNT studied).

The above inferences ascertain the fact that the molecular vacancy defects in SWCNT are one potential cause for the reduction of Young's modulus of the EPON-SWCNT nanocomposite.

The reduction in Young's modulus between the pure CNT composite and the composite with two defects in the CNT was in the range of 13-18% while that between the pure CNT composite and the composite with four defects in the CNT was 21-30%. This is presented in Table 3.

Table 3. Percentage reduction in Young's modulus with the introduction of carbon vacancy defects in CNT

CNT weight %	Young's Modulus [GPa]	
(No defect)		
	% reduction with 2 defects in composite	% reduction with 4 defects in composite
11.87	17.9 (2.78% defects in CNT)	29.6 (5.56% defects in CNT)
10.69	15.7 (2.08% defects in CNT)	25.1 (4.17% defects in CNT)
8.24	13.6 (2.08% defects in CNT)	21.9 (4.17% defects in CNT)

Table 3 can be interpreted as follows:

- For the 11.87% SWCNT composite system, introduction of 2.78% (equivalent to 2 carbon vacancy defects) of defects into the SWCNT resulted in 17.9% overall reduction in the Young's modulus of the composite. The introduction of 5.56% of defects in the SWCNT (corresponding to 4 carbon vacancy defects) resulted in 29.6% reduction in the Young's modulus of the composite.

- For the 10.67% SWCNT composite system, introduction of 2.08% defects (corresponding to 2 carbon vacancy defects in the SWCNT) resulted in 15.7% overall reduction in the Young's modulus of the composite while introduction of 4.17% of defects (corresponding to 4 carbon vacancy defects) in the SWCNT resulted in 25.1% reduction in the Young's modulus of the composite.
- For the 8.24% SWCNT composite, introduction of 2.08% of defects into the SWCNT resulted in 13.6% overall reduction in the Young's modulus of the composite while introduction of 4.17% of defects into the CNT resulted in 21.9% reduction in the Young's modulus of the composite.

Summary and conclusions

In this work, the effect of carbon vacancy defects on the mechanical properties (in particular Young's Modulus) of SWCNT-EPON 862-DETDA nanocomposite was investigated with MD simulations employing Materials Studio and Accelrys.

Three epoxy – SWCNT molecular model configurations with different weight percentages of SWCNT was employed in this study. These molecular models had SWCNT weight percentages ranging between 7% and 12%. Two types of carbon vacancy defects were incorporated into the defective SWCNT models; 2 defects and 4 defects. After creation, minimization, and equilibration of the different molecular models; MD simulations were conducted following a simulated annealing process with temperatures ranging from 498 °K to 298 °K in steps of 10 °K. Dynamic simulations were conducted for 200 ps (200,000 fs) with a time step of 1 fs at each simulated annealing temperature studied. The final molecular structure of each annealing temperature step was used as the starting molecular structure of the next annealing temperature step. The average density was obtained at each simulated annealing temperature step. At 298 °K, 10 trajectories were saved at equal time intervals (20 ps) and employed in the mechanical property estimations. The models with two defects (2.08-2.78% defects from the reduction in carbon atoms due to their removal to create SWCNT carbon vacancy defects) showed a reduction in Young's modulus between 13-18% when compared with the non-defective SWCNT – EPON composite models. The models with four defects (4.17-5.56% defects) showed a 21-30% reduction in the Young's modulus compared to pure, non-defective SWCNT – composite molecular models. The influence of SWCNT defects (due to carbon vacancy defects in the present study) reducing the Young's modulus could potentially be a contributor to the disparity seen between the MD modeling results and experimental data cited in the literature.

REFERENCES

- [1] Liu H., Brinson L. C., *A hybrid numerical-analytical method for modeling the viscoelastic properties of polymer nanocomposites*, Journal of Applied Mechanics, Volume 73, (2006) 758-762.
- [2] Miyagawa H., Rich M., Drzal T. L., *Thermophysical properties of anhydride-cured epoxy/nano-clay composites*, Polymer Composites, Volume 26, Issue 1, (2005) 42-51.
- [3] Dutra R., Soares B., Campos E., Silva G., *Hybrid composites based on polypropylene and carbon fiber and epoxy matrix*, Polymer, Volume 41, (2000) 3841-3849.
- [4] Zhou Y. X., Wu P. X., Cheng Z. Y., Ingram J., Jeelani S., *Improvement in electrical, thermal and mechanical properties of epoxy by filling carbon nanotube*, Express Polymer Letters, Volume 2, Number 1, (2008) 40-48.
- [5] Demczyk B. G., Wang Y. M., Cumings J., Hetman M., Han W., Zettl A., Ritchie R. O. *Direct mechanical measurement of the tensile strength and elastic modulus of multi-walled carbon nanotubes*. Materials Science and Engineering A, Volume 334, (2002) 173-178.
- [6] Ning H., Zen M., Cheng Y., Go Y., Hsiao F., Toshiyuki H., *The electrical properties of polymer nanocomposites with carbon nanotube fillers*, Nanotechnology, Volume 19, (2008) 215701.
- [7] Yue H., Elliot J., *Molecular dynamics simulations of the elastic properties of polymer/carbon nanotube composites*, Computational Materials Science, Volume 39, (2007) 315-323.
- [8] Zhu R., Pan E., Roy A. K., *Molecular dynamics study of the stress-strain behavior of carbon-nanotube reinforced EPON 862 composites*, Materials Science and Engineering A, Volume 447, (2007) 51-57.
- [9] Gou J., Minaie B., Wang B., Liang Z., Zhang C., *Computational and experimental study of interfacial bonding of single-walled nanotube reinforced composites*, Computational Materials Science, Volume 31, (2004) 225-236.
- [10] Komuves F., *Prediction of mechanical properties of EPON 862 (DGEBA) cross-linked with curing agent W (DETA) and (6,6) SWCNT using MD simulations*, PhD Dissertation, Mechanical Engineering 2009.

- [11] Tack J. L., Ford D. M., *Thermodynamic and mechanical properties of epoxy resin DGEBF cross-linked with DETDA by molecular dynamics*, Journal of Molecular Graphics and Modeling, Volume 26, (2008) 1269-1275.
- [12] Davis D., Klosterman, Kelkar A., Bolick R., Mohan R., *Composite laminate structures for mechanical and functional processes*, Quarterly Report, April 2009-June 2009.
- [13] Li Z., Wang C. Y., Ke S. H., Yang W., *First principles study for transport properties of defective carbon nanotubes with oxygen adsorption*, The European Physical Journal B, Volume 69, (2009) 375-382.
- [14] Mielke S. L., Troya D., Zhang S., Li J. L., Xiao S., Car R., Ruoff R. S., Schatz G. C., Belytschko T., *The role of vacancy defects and holes in the fracture of carbon nanotube*. Chemical Physics Letters, Volume 390, (2004) 413-420.
- [15] Mielke L. S., Zhang S., Khare R., Troya D., Ruoff R. S., Schatz G. C., Belytschko T., *Mechanics of defects in carbon nanotubes: atomistic and multiscale simulations*, Physical Review B, Volume 71, (2005) 115403.
- [16] Chico L., Crespi V. L., Benedict L. X., Louie S. G., Cohen M. L., *Pure carbon nanoscale devices: nanotube heterojunctions*, Physical Review Letters, Volume 76, Number 6, (2006) 971-974.
- [17] Fefey E., Mohan R., Kelkar, A., *Computational study of the effect of carbon vacancy defects on the Young's modulus of (6, 6) Single Wall Carbon Nanotube*, Material Science and Engineering – part B, 176(9), 2001.
- [18] Park Y., Lahaye R. J., Lee Y., *Adsorption of Pt on defective carbon nanotube walls: a DFT approach*. Computer Physics Communications, Volume 177, Issues 1-2, (2007) 46.
- [19] Liu W.K., Karpov E. G., Zhang S., Park H. S., *An introduction to computational nanomechanics and materials*, Computer Methods in Applied Mechanics and Engineering, Volume 193, (2004) 1529-1578.
- [20] Sun H., *The COMPASS force field: parameterization and validation for phosphazenes*, Computational and Theoretical Polymer Science, Volume 8, Number 1/2, (1998) 229-246.
- [21] Hu Y., Sinnott B. S., *Constant temperature molecular dynamics simulations of energetic particle–solid collisions: comparison of temperature control methods* Journal of Computational Physics, Volume 200, Issue 2, (2004) 251-266.

- [22] Andersen C. H., *Molecular dynamics simulations at constant pressure and/or temperature*. Journal of Chemical Physics, Volume 2, Number 4, (1980) 2384-2393.
- [23] Brooks S. P., Morgan B. J. T., *Optimization using simulated annealing*, The Statistician, Volume 44, Number 2, (1995), 241-257.

A-1-2: Modeling and Experimental Investigation on the Effect of Interlaminar Nanofiber Layers on the Delamination Behavior in an Epoxy Fiber Glass Composite

Authors: R. Mohan, A. Kelkar, N. Chinnanavar, S. Shendokar, North Carolina A&T State University

M.S. Thesis work of N. Chinnanavar

Abstract

Delamination is one of the important failure mechanisms in composite materials. Several methods such as stitching of fiber plies, self-healing polymer materials, and interface reinforcements have been developed, investigated and employed over the years to improve the delamination characteristics. The usage of interface material layers (in particular, sub-micron and nano level materials) has also been recently investigated. This study focuses on the addition of electrospun nano fiber interface layers between the traditional composite laminates and its effect on the delamination characteristics in an epoxy-fiber glass composite system. Electrospun glass nano fiber layers formed with TEOS (Tetra Ethyl Ortho Silicate) sol gel system are used as interface layers. Delamination characteristics of the composite with and without electrospun fiber interface layers are studied using double cantilever beam (DCB) tests. The experimental characterization showed that addition of nano fiber layers provided consistent improvements in the Mode-I fracture toughness values. Finite element modeling of the crack growth and delamination failure with and without the nano fiber layers are studied and compared. The Mode-I fracture toughness values from the finite element modeling are compared with the experimental data.

1 Introduction

High specific strength, ability to be tailored with desired directional properties along with integrability of cores and stiffeners easily; adaptability for complex shapes, corrosion resistance, dimensional and hygro-thermal stability with excellent fatigue performance and low specific cost are some of the prominent properties of composite materials that influences their widespread application in the areas of aerospace, naval, automobile, wind energy, bridge and sports goods industry. In spite of these advantages, there are integral challenges with composite material development that arise due to inherent anisotropic nature of these laminated structures. A laminated composite structure consists of fiber laminas which could be unidirectional, cross-ply or multi-directional embedded in polymer matrix resin. While the fibers are the load carrying members, the resin matrix is a load transfer member. Over the period of last 30 years variety of fibers made of glass, organic (aramid), ceramic and carbon fibers have demonstrated effective reinforcement to provide tailored properties of composite parts. Due to their good chemical and

thermal stability thermoset resins in particular epoxies are predominantly used as matrix for many of the applications listed above.

As the strength of the fiber is much higher than the polymer matrix, failures of composites is initiated in the matrix. Fiber-Polymer matrix composites have very high anisotropy; hence the studies of failure mechanisms are much more complex. Complexity increases as failure of composite parts is also dependent on the loading conditions, shape (geometry) and the properties of its constituents. Invariably failure of composites at the micro level is in the form of damage initiation, which over the life of a composite, is dispersed at random locations. When the density of micro damage increases, there is a tendency towards coalescence leading into catastrophic fracture [1-2]. The three regions in a composite that can experience damage leading to the fracture are fiber, matrix or interface. Consequently, the failure modes identified in composites are (i) Matrix Cracking, (ii) Interface Cracking, (iii) Delamination and (iv) Fiber breaking. Figure 1 shows schematically various failure modes that could occur in laminated fiber reinforced polymer matrix composites.

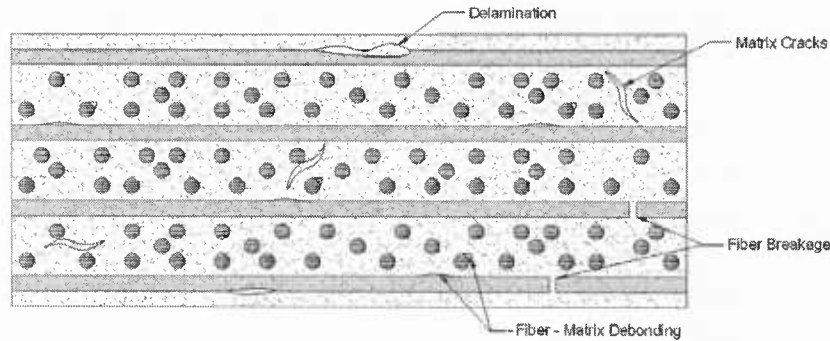


Figure 1: Failure Modes in Composites

One of the weakest and most frequently encountered failure modes of fiber glass composite laminates is interlaminar delamination. This is also referred to as interlaminar cracking that can cause severe loss of stiffness, which can propagate into catastrophic failure by means of splitting of plies in a laminated composite. Delamination failure is complex due to geometric and material discontinuity coupled with variety of loading that causes growth of delamination [3].

Various methods have been adopted to improve resistance to delamination. One of the earliest attempts to improve resistance to delamination by optimizing stacking sequence was by Pagano and Pipes [4]. They predicted detailed stacking sequence and specific layer orientations to suppress damage growth under uniaxial static and fatigue loading. It was inferred that interlaminar normal and shear stresses lead to coalescence of micro-cracks resulting in the strength degradation due to delamination. It was proved experimentally, that the optimized stacking sequence resulted in better strength of composite laminates. Matrix toughening is another method that has proved to be effective to resist delamination. Various types of additives like alumina nanoparticles [5] and CNT's [6] have been demonstrated to increase fracture

toughness. There exists a stress concentration at the free edges of composite laminate, which leads into delamination. To suppress this stress concentration various methods of edge design have been studied in the past [7]. Stitching, braiding and knitting techniques have been adopted by many researchers to improve out of plane properties. While out of plane properties were improved using these techniques, in some cases drastically, there was substantial degradation in the in-plane properties of the composites. Interleaving is one of the prominent mechanisms adopted and studied to improve resistance to delamination in the recent past [8-10]. These methods while successful resulted in pitfalls which led to either degradation of some of the properties or an increase in weight and/or increase in the cost of composite material/manufacturing. In present study, performance and effect of interleaved electrospun nanofibers in a glass fiber prepreg composite is analyzed experimentally and compared with a simplified finite element modeling approach for Mode I fracture toughness.

There are several of finite element based approaches developed to analyze mode I fracture toughness. Virtual Crack closure technique (VCCT) is one such approach introduced by Ronald Krueger and Andrzej Leski [11-12]. VCCT is based on Irwin's crack closure integral, which assumes that the energy ΔE released during infinitesimal crack growth from a to $a + \Delta a$ is equivalent to the energy required to close the same crack. Bonhomme [13] has extended the study on the VCCT method in his research to introduce the two step extension method. In the two step method, the crack path is modeled with a pair of nodes coincident at the same location. The fracture toughness is calculated in two steps compared to one step in VCCT. In the first step, the forces at crack tip are calculated; and then an imposed displacement is applied in conjunction with the release of coupled degrees of freedoms of nodes. J-Integral is a method to calculate strain energy release rate per unit fracture surface area. This was developed by Cherepanov [14] and later modified by Jim Rice [15]. While these analytical methods are proved to be suitable for Mode I fracture toughness assessment via finite element modeling, they are complex in their implementation and expensive in terms of computational resource.

In this work, we evaluate the applicability and performance of a simplified finite element modeling approach proposed to analyze mode I fracture toughness. Mode I fracture toughness values from the finite element modeling are compared to the experimentally determined values. The material system used was LTM45EL/7725 pre-pregs with layers of tetra ethyl ortho-silicate (TEOS) nanofibers produced by electrospinning at the critical interface of a double cantilever beam (DCB) specimen. Finite element modeling results are compared for the DCB specimens those were made and tested using ASTM D 5528 standard.

2 Experimental Investigations

This section briefly discusses the manufacturing of TEOS electrospun nanofibers and DCB specimen for Mode I fracture toughness characterization using ASTM D 5528 standard procedure. These experimental investigations were conducted as a part of research effort on integrated composite technologies through a project on Center on Nano Science and Materials at North Carolina A&T State University.

2.1 Electrospinning of TEOS Nanofibers

Electrospinning is a simple and versatile process to generate ultra-thin fibers from a variety of polymer, ceramic or composite solutions [16-17]. The fundamental four components associated with the electrospinning process as seen in the Figure 2 (a). In the electrospinning process, a solution droplet is fed to the spinneret tip at a controlled rate using a programmable dispensing pump. The dispensing pump is a Model NE-1000 Multi-Phaser supplied by New Era Pump Systems Inc., and has the capacity of holding a syringe up to 50 mm in diameter. The pump can dispense solution over a wide volume range of 0.1 ml per min. to 10 ml per min.

The solution droplet at the tip of the spinneret is acted upon by electro-hydrodynamic forces. The electrical forces are due to the potential difference applied between the spinneret and the collector plate. The spinneret is kept at a positive potential and the collector plate is usually kept grounded. A FC Series, 120 Watt Regulated High Voltage DC Power Supply supplied by Glassman High Voltage, Inc., maintained a voltage of 18kV between the spinneret and the collector plate. Due to this applied potential difference, the solution droplet at the tip of the spinneret acquires positive charge on the surface. The hydro-dynamic forces are due to the surface tension of the liquid solution. The solution droplet is attracted to the collector plate and forms a 45° semi-angle at the tip. The formed shape is called a “Taylor Cone” [18]. When the viscosity of the solution is sufficient to provide stringiness, there is an elongation of the droplet into a jet, which under the action of whipping and “Bending Instability” [19] forms fibers in the range of 3nm to 1µm in diameter depending on the solution properties.

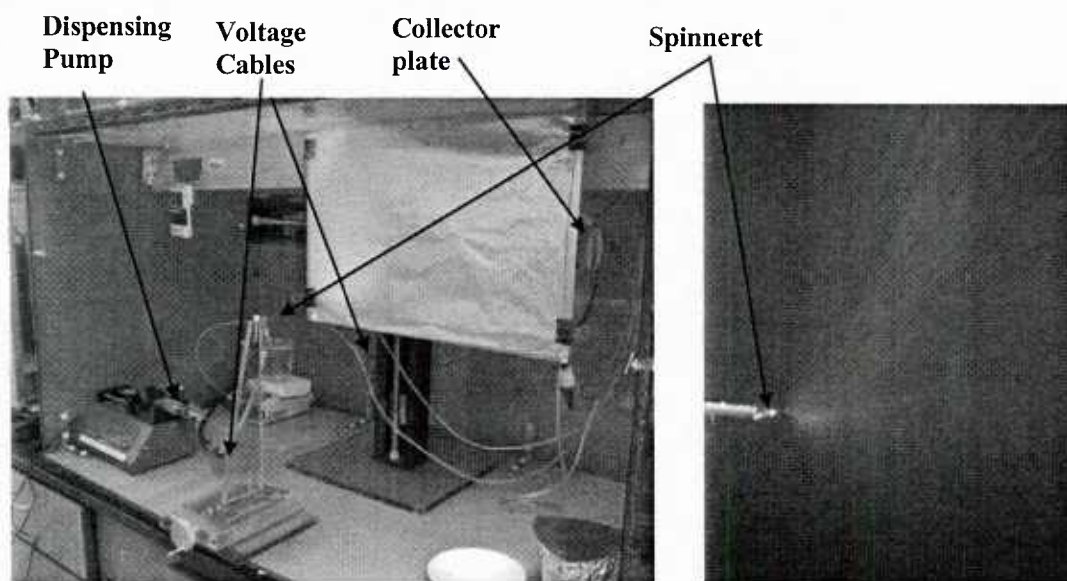


Figure 2: a) Electrospinning Setup,

b) Electrospinning Fiber Jet

Figure 2 (b) illustrates a droplet extending into a jet and then into an instability region. As the droplet is stretched into a fiber and deposited onto the collector plate, there is an evaporation of solvents. The deposition of fibers onto the collector plate is a random dispersion. The thickness of this deposition is controlled using a steady motion of collector plate using the X-Y Velmex Slides. The length of the deposited fibers can be 10 to 100 times to that of the fiber diameter. To produce fibers less than 500 nm in diameter consistently, the electrospinning parameters are experimentally found to be: 1 ml/hr rate of dispensing, 18 kV potential difference, a 80 mm distance between the spinneret and the grounded collector plate and solution viscosity of 100-200 centipoises. It was observed that, if the TEOS solution at the time of electrospinning contains an excess of solvents, the formed random nanofiber mats evaporate off these solvents at room temperature and generate cracks. Hence, the TEOS random nanofiber sheets were kept at a normal atmospheric room temperature for about two days for natural evaporation of the solvents to assess the consistency of the electrospun nanofibers. The morphology of the electrospun fibers can be studied using a Scanning Electron Microscope (SEM). Figure 3 illustrates micrograph of the TEOS electrospun fibers captured using a Hitachi S-3000 N SEM at 6000x magnification.

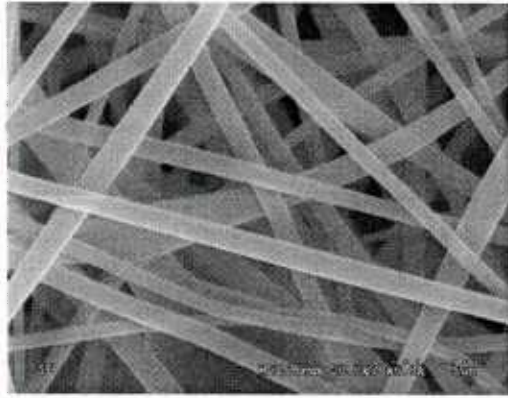
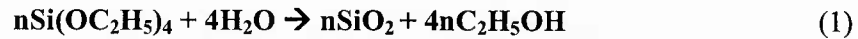


Figure 3: SEM Micrograph of TEOS Electrospun Nanofiber

The spinnable TEOS solution is obtained by hydrolysis and poly-condensation during the aging process of TEOS as specified by the following chemical reaction [20]. Ethanol is used as solvent for hydrolysis and HCL is used as catalyst to accelerate cross linking during aging process.



2.2. Mode I Characterization of TEOS Interleaved DCB Specimen

One of the most critical tasks for conducting DCB tests is to make DCB specimen. It is critical because if the specification of ASTM standard is to be achieved, significant control is required while manufacturing the composite panel. The geometry specified for a DCB coupon specimen is as shown in Figure 4 with the range for the dimensions within which DCB specimen should be made. Final consolidation of the DCB specimens was completed under atmospheric vacuum and temperature.

While laying up composite panel for DCB specimens, Teflon film is laid up at the mid-plane. This is above the fifth ply of a ten ply E-glass fiber composite. The length of the Teflon film should be about 4 inches while laying up the plies; with 1 inch Teflon film for the trimming allowance and 1 inch for the piano hinge tab length. This ensures a 50.8 mm (2.0 in) initial crack length in the DCB specimen. The thickness of the non-adhesive insert for the initial crack should not be greater than 13 microns (0.0005 inch). For resin requiring a temperature less than 177 °C (350 °F), polytetrafluoroethylene (PTFE) is recommended. For resins requiring curing above this temperature, a polyimide film is preferred.

The specimen dimension as recommended in the ASTM standard requires a 127 mm (5 inches) specimen length, but it is recommended to have at least 50.8 mm (2 inches) more in length, so as

to make a total of 177.8 mm (7 inches). Out of this 25.4 mm (1 inch) will be for the bonding of piano hinges, 50.8 mm (2 inch) initial crack length and the crack growth observed up to 50.8 mm (2 inch) with additional 50.8 mm (2 inch) remaining as the end tolerance. The Mode I crack opening is independent on the width of the specimen with the suggested width of specimen of about 0.8 inch to 1.0 inch. The thickness recommended is 3-5 mm (0.12 - 0.2 inches). The detailed procedure for bonding piano hinges and generating markings on “White-out” applied on one side of the DCB specimen is elaborated in the standard. The marking which start from the tip of the non-adhesive insert has the first five graduation spaced at 1.02 mm (0.04 inches) and the remaining graduations are spaced at 5.08 mm (0.2 inch).

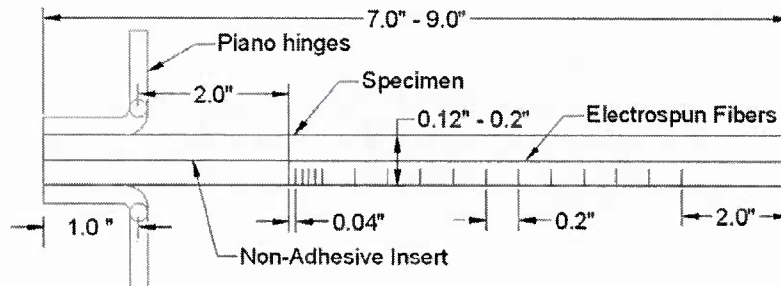


Figure 4: ASTM DCB Specimen Configuration

A Modified Beam Theory (MBT) method which gives most conservative values was used to compute the Mode I fracture toughness (G_{IC}) values. MBT calculates G_{IC} as follows

$$G_{IC} \equiv \frac{3 \times P \times \delta}{2 \times b \times a} \quad (2)$$

where,

P = Load N (lbf),

δ = Load point displacement mm (in),

b = Specimen width mm (in), and

a = Crack (delamination) length mm (in).

G_{IC} = Mode I Interlaminar Fracture Toughness J/m^2 (lbf/in²)

Table 1 presents the mode I fracture toughness values obtained experimentally for the composite DCB specimen with and without electro-spun layers.

Specimen	Neat (J/m^2)	Espun 1.0 gm (J/m^2)
1	335.87	646.76
2	419.26	578.00

3	412.34	655.64
4	332.56	801.76
Avg. G_{IC}	375.00	670.54
Std. Dev.	47.20	94.11

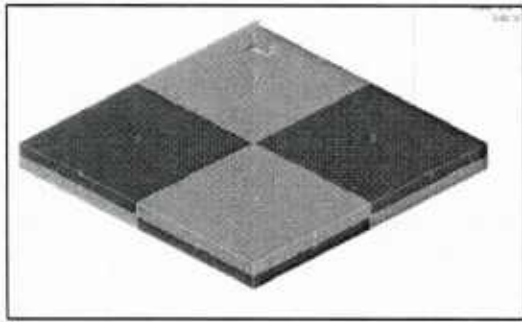
Table 1: Comparison of average G_{IC} values for Prepreg composites with and without electrospun interface layer

3 Finite Element Modeling and Methodology for G_{IC} Computation

3.1. Modeling Geometry of DCB Specimen

The fundamental concept for the finite element modeling and analysis of DCB specimen currently is based on the incremental deformation and damage progression of elements. The continuous deformation is provided by incremental escalation of load values to a point where few elements in the model exceeds the failure stress. Those elements which experience stress level higher than the failure stress are substituted with weak material properties so as to identify the “Failed” elements. Damage progression is simulated iteratively by load increment and identifying and updating of failed elements.

The finite modeling of the composite laminate is conducted using a mosaic geometry for the warp and weft elements of woven fabric [21]. In a modeled laminate, mosaic geometry is interpreted as one mosaic cell representing warp (0 degree) and another mosaic representing weft (90 degree). Eight such mosaic elements are placed in orthogonal array as shown in for a plain weave fabric unit cell model. Dimensions of unit cell conforms the warp and weft tow geometry of the actual prepreg material LTM45EL/7725 modeled currently as plain weave. Each unit cell consists of 4 warp elements and 4 weft elements as shown in Figure 5. The unit cell is repeated along the length and width to get the required specimen dimensions to form a single lamina layer. This layer is repeated along the thickness direction so as to be equivalent to the actual dimension of DCB specimen as that was used in the experiments.



	Length X (inch)	Thickness Y(inch)	Width Z(inch)
Warp	0.0787	0.0098	0.0787
Weft	0.0787	0.0098	0.0787

Figure 5: Unit cell showing warp (0 Degree) and weft (90 degree) and the dimensions

3.2. Boundary Conditions and Loads

Due to symmetry, only one half the thickness of the DCB specimen geometry was modeled. Symmetric boundary conditions were applied as defined to be zero displacements in y and z directions from the point of crack tip for the bottom layered nodes as shown in Figure 6. The load P was applied during finite element modeling analysis to conform to the load application during the experiments as per ASTM standards at one inch from the edge of the DCB model geometry and at two inches from the crack tip. Figure 7 shows the position of load application in the FEM model.

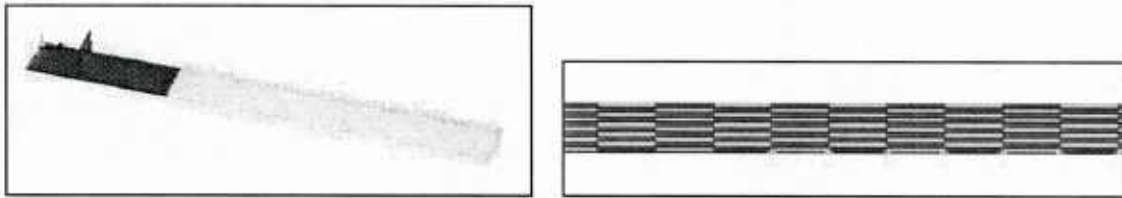


Figure 6: Constraints and load applied in the finite element model

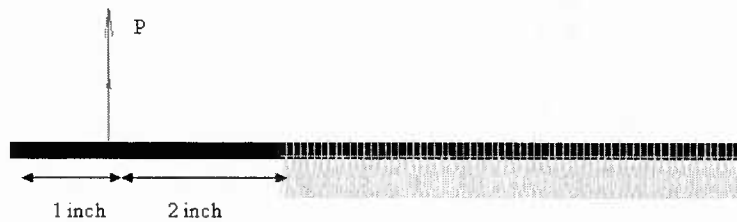


Figure 7: Load P applied in the DCB finite element model

3.3. Failure Criteria and Degradation of Elements

During static finite element analysis of the crack propagation in the DCB specimen, load was increased incrementally through distinct consecutive finite element analysis, and the corresponding Von Mises stresses along x -direction near the crack tip were examined. The incremental load increase was continued until the elements in the vicinity of crack tip failed. The criterion for the failure of the elements was set to be the yield point of the resin, which is about 7,500 psi [22]. Once the stresses in the elements in the vicinity of the crack tip reaches 7,500 psi or more, they were considered to have failed and degradation of these elements was done. The failed elements still possess a reduced load carrying capacity and are still involved in the failure phenomena. This behavior of failed elements is implemented in the finite element modeling analysis by degrading the properties of the failed elements. Degradation of elements is also known as killing of elements, in which the elastic moduli of failed elements are reduced to a negligible value of 100 psi in the present work, and a poison's ratio of 0.01. The modulus is not reduced to zero to avoid numerical difficulties which occur in finite element computations. The crack growth phenomenon in the finite element analysis was implemented by the method of degrading the failed elements and removing the constraints associated with these elements so that they will have a very negligible effect on the subsequent analysis for the crack growth. This method of removing the constraints was utilized to emulate the crack growth the finite element models.

3.4. Finite Element Modeling Results for DCB Characterization

The composite material properties used in the finite element modeling were calculated using rule of mixtures. To determine equivalent composite properties, baseline modulus of fabric 7725 and Epon 862 were used with a 60% fiber volume fraction. The electrospun fiber layers were added as an additional homogeneous material layer in the finite element model and were defined with the equivalent fiber properties. The conditions for the propagation and growth in the DCB specimen finite element models for the electrospun fiber interface layer composites are taken to be the same as that of the neat composite DCB specimens. Mode I fracture toughness values were computed according to the modified beam theory as specified in ASTM 5528 standard [23] with the load and displacement values obtained from the finite element analysis. The static, linear finite element analysis was performed using the commercial ANSYS finite element analysis software. The load-crack growth characteristics and the Mode-I fracture toughness values from the finite element analysis are compared. Mode I fracture toughness values determined experimentally from one set of experiments and from the present finite element modeling are presented in Table 3.

Fracture toughness G_{IC} (J/m ²)	Finite Element Analysis (J/m ²)	Average value from experimental test (J/m ²)
--	--	---

Neat Composite	429.25	375.23
Electro-spun Interface Composite	566.76	670.54

Table 3: Mode I Fracture toughness G_{IC} by finite element method and experimental characterization

4. Concluding Remarks

Electrospun nanofiber interface layers in woven fiber composites provide an effective way of integration of nanomaterial systems in fiber composites and have shown to improve the delaminating mode I fracture toughness characteristics. A simplified finite element modeling approach was investigated for the modeling and analysis of the crack propagation in the composite DCB specimen. The experimental Mode I fracture toughness values and the finite element modeling analysis both based on modified beam theory are compared. The mode I fracture toughness G_{IC} obtained by finite element analysis for neat E-glass laminate were over predicted by 14 percent when compared to the mean value of the experimental data and under predicted by 15 percent over the mean experimental value in the case of E-glass laminate with an electro-spun interface layer. The maximum value of experimental results differed only by 8 J/m² when compared with finite element analysis result for E-glass prepreg laminate without electro-spun interface, while the finite element analysis results compared with E-glass electro-spun interface layer varied by only 9.75 J/m² from the minimum value of the experimental data. The mode I fracture toughness from the finite element modeling are in agreement and consistent within the experimental range values. The present finite element modeling analysis is based on a linear elastic model and a simplified approach of crack propagation by degradation. Even with the simplified crack propagation model and linear elastic analysis in the finite element modeling, the present simplified finite element modeling approach captures the crack propagation behavior effectively and provides a computationally efficient modeling approach for delamination.

References

- [1] Talreja R.: Transverse cracking and stiffness reduction in composite laminates, *Journal of Composite Materials*, **19**, 355-375 (1985).
- [2] Talreja R.: A continuum mechanics characterization of damage in composite materials, *Proceedings of the Royal Society of London. Mathematical and Physical Sciences*, **399** (1817), 195-216, (1985).
- [3] Wang S.: Edge delamination in angle ply composite laminates, University of Illinois, Urbana Champaign, NASA Report, (1995).
- [4] Pagano N., Pipes R.: The influence of stacking sequence on laminate strength, *Journal of Composite Materials*, **5**, 50-57, (1971).

- [5] Akinyede O., Mohan R.V., Kelkar A. D., Sankar J.: Static and fatigue behavior of epoxy/fiberglass composites hybridized with alumina nanoparticles, *Journal of Composite Materials*, **43**(7), 769-781, (2009).
- [6] Sadeghian R., Gangireddy S., Minaie B., Hsiao K.: Manufacturing carbon nanofibers toughened polyester/glass fiber composites using vacuum assisted resin transfer molding for enhancing the mode-I delamination resistance, *Composites: Part A*, **37**, 1787-1795, (2006),.
- [7] Wu X., Dzenis Y.: Experimental determination of probabilistic edge-delamination strength of a graphite-fiber/epoxy composite, *Composite Structures*, **70**(1), 100-108. (2005).
- [8] Kim J., Reneker D.: Mechanical properties of composites using ultrafine electrospun fibers, *Polymer Composites*, **20**(1), 124-131, (1999).
- [9] Jiang W., Tjong S., Chu P., Li R., Kim J., Mai Y.: Interlaminar fracture properties of carbon fiber epoxy composites interleaved with polyethylene terephthalate films, *Polymers and Polymer Composites*, **9**(2), 141-145, (2001).
- [10] Liu L., Liang Y., Xu G., Zhang H., Huang Z.: Mode I interlaminar fracture of composite laminates incorporating with ultra-thin fibrous sheets, *Journal of Reinforced Plastics and Composites*, Article in Press, **27**, 1147-1162, (2008).
- [11] Krueger R., O'Brien T.: A shell/3d modeling technique for the analysis of delaminated composite laminates, *Composites Part A: Applied Science and Manufacturing*, **32**, 25-44, (2001).
- [12] Leski A.: Implementation of the virtual crack closure technique in engineering FE calculations, *Finite Elements in Analysis and Design*, **43**, 261-268, (2007).
- [13] Bonhomme J. et al.: Numerical and experimental validation of computational models for mode I composite fracture failure, *Computational Materials Science*, **45**, 993-998, (2009).
- [14] Cherepanov G.: Crack propagation in continuous media, *Journal of Applied Mathematics and Mechanics*, **31**, 503-512, (1967).
- [15] Rice J.: A path independent integral and the approximate analysis of strain concentration by notches and cracks, *Journal of Applied Mechanics*, Vol. 35, pp. 379-386, 1968.
- [16] Formhals A.: Process and apparatus for preparing artificial threads, US patent 1,975,504, USA, (1934).
- [17] Doshi J., Reneker D., Electrospinning process and applications of electrospun fibers, *Journal of Electrostatics*, **35**, 151-160, (1995).
- [18] Taylor G.: Disintegration of water drops in an electric field, *Proceedings of Royal Society of London, Ser A*, **280**, 383-397, (1964).
- [19] Yarin A., Koombhongse S., Reneker D.: Bending instability in electrospinning of nanofibers, *Journal of Applied Physics*, **89** (5), 3018-3026, (2001).
- [20] Sakka S., Kamiya K.: The sol gel transition in the hydrolysis of metal alkoxides in the relation to the formation of glass fibers and films, *Journal of Non-Crystalline Solids*, **48**, 31-46, (1982).

- [21] LTLM45EL Resin Data; 7725 Fabric data: <http://www.advanced-composites.co.uk>
- [22] <http://www.hexion.com/Products/TechnicalDataSheet.aspx?id=3950>
- [23] ASTM D 5528.: Standard test method for mode I interlaminar fracture toughness of unidirectional fiber-reinforced polymer matrix composites, ASTM International, USA.

A-2 Computational multi-scale deformation behavior in metallic and non-metallic systems

Research activities and technical approach in this area during the project period focused on the deformation behavior of nanoscale material systems with applications to tensile, flexural, and crack propagation.

A-2-1 Molecular Dynamics Modeling of tensile, flexural and crack propagation in metallic systems

Authors: R. Mohan, Y. Purohit, Y. Liang, North Carolina A&T State University

Published Journal Article: *Journal of Computational and Theoretical Nanoscience*, Vol. 9, Pages 1-13, 2012.

ABSTRACT

Nanomechanics is an evolving field that investigates the mechanical properties, deformation behavior and characteristics of nanoscale structures. Due to the smaller lengths at the nano level, principles of mechanics are employed in conjunction with interatomic potentials, molecular forces and molecular dynamics. This paper highlights the underlying principles and discusses the tensile and flexural deformation of Nickel nanowires; and dynamic crack propagation in nanoscale Nickel and Nickel-Aluminum bimetal interface.

The tensile deformation behavior analysis indicates that Young's Modulus was independent of the cross sectional area of the nanowire, and the strain rate. The flexural deformation and vibration behavior indicates that the frequency of the vibrations as computed from time displacement deformation behavior of the molecular configurations of the Nickel nanowire beams are independent of the magnitude of external loading, and is consistent with the classical beam theory.

The dynamic crack propagation behavior in a Nickel single crystal and a Nickel-Aluminum bimetal interface are investigated. The propagation mechanisms and fracture behavior in Ni are compared with such behavior in Ni-Al nanoscale bimetallic layer that initiates and propagates from Ni towards the Ni-Al bimetal interface. Our results for Ni show an initial brittle crack propagation followed by a roughening of the crack surfaces at one-third of the Rayleigh wave speed. In Ni-Al, the crack surfaces initially grow brittle. Two regimes of crack propagation velocities were observed in this case with crack getting decelerated as it nears the interface.

Further dynamic analysis of the crack propagation indicated a cease in the crack propagation in Ni due to a brittle to ductile transition. In Ni-Al bimetal interface system, as the crack approaches the interface, a process zone representing local disorder at the crack tip was observed to start growing and interacting with interfacial defects that eventually results in a blunting of the crack tip.

INTRODUCTION

Deformation behaviors of nanoscale metallic systems under mechanical loading conditions have received considerable attention in the recent years. For example, in applications such as nanoelectronics and nano-optoelectronics¹, the extraordinary mechanical strength along with the small dimensions for the efficient transport of electrons of metallic nanowires have shown great potential for the minimization of electronic devices. These metallic nanowires also show potential for applications in electronic packaging, nanoelectronic and nanomechanical devices. The structural strength and the stability under mechanical and thermal loading conditions of such nanowires is however a significant issue. The deformation behavior of these nanowires under different mechanical loads (for e.g., tensile, bending) is poorly known. Experimental investigations of these behaviors are impractical due to their size and the complications of applying these loading conditions via nano load cells within high resolution microscope systems. Continuum mechanics based approaches generally treat the small cross-sectional area configurations of these nanowires to be one-dimensional, where the cross-sectional effects are taken to be negligible. However, at the atomistic level, the mechanical deformation and the failure characteristics are inherently three-dimensional; depend upon the atomistic level interactions and require analysis methodologies that effectively emulate the three dimensional atomistic level characteristics.

Associated with mechanical deformation of the material systems are their fracture and failure. Even at structural macro scale, the homogeneous, macroscopic (continuum) behavior is governed by the physical processes that occur at the heterogeneous microscopic and sub-microscopic length scales. For example, most metallic materials at macro scale consist of polycrystalline aggregate of heterogeneous grains at the fine scale. The mechanisms of fracture and the crack propagation not only depend upon the type of loading but also upon the type of defects such as grain boundaries present as well as the physical interactions of dislocations in the microscale grain boundaries. The forming dislocations in the single grain of the metallic material may also depend on the heterogeneous interactions between the lattices of metal atoms at the atomistic levels.

Continuum-based theories of fracture mechanics provide a variety of energy and force criteria to model and predict the critical conditions for the onset and further growth of statically or quasi-statically loaded stationary cracks. The continuum theories have led to the development of a

detailed understanding of the mechanics of fracture. Despite their valuable contributions, continuum modeling does not provide atomistic level details at nanoscale dimensions, interface structures and properties, internal stress and energy distribution, dislocation nucleation and motion; crack propagation and its interaction with interfaces in metallic composites to include the effect of structures and processes that become important at nano scales. The strong constraint of the small length scale on the crack behavior and dislocation activities at the nanometer scale also give rise to deformation and failure mechanisms that differ significantly from the bulk metals. Moreover, the interaction of cracks and dislocations with the interfaces becomes the controlling parameter of plasticity in these systems. The detailed understanding of this problem includes nucleation of dislocations at the crack tip, creation of dislocations at interfaces, transmission of dislocations through interfaces and emission of dislocations from the interfaces. This is only possible through three-dimensional atomistic level characteristics of such material nanoscale material systems.

Nanomechanics is an evolving field that investigates the mechanical properties, deformation behavior and characteristics of nanoscale structures. Due to the extremely smaller lengths at the nano level and to capture the three-dimensional atomistic deformation characteristics of nanoscale material systems, principles of mechanics are employed in conjunction with interatomic potentials capturing the molecular forces, atomistic level interactions and molecular dynamics; and offer a potential to understand the associated deformation behavior at nanoscale. The computational modeling of nanoscale deformation behavior employs computational techniques based on molecular dynamics simulations that couple the principles of mechanics with molecular forces and interatomic potential providing an effective methodology. The present paper focuses on the computational modeling of the deformation behavior in nanoscale material systems with applications to tensile, flexural and crack propagation in nano scale metallic systems. In particular, this paper discusses the tensile and flexural deformation of Nickel nanowires; and nanoscale dynamic crack propagation in Nickel and Nickel-Aluminum bimetal interface. The basic principles associated with the modeling of deformation behavior in nanoscale material systems are briefly highlighted first. This is followed by the discussions on the tensile and flexural deformation behavior in Nickel nanowires and dynamic fracture in a Nickel-Aluminum nano scale bimetallic interface.

NANOMECHANICS AND MOLECULAR DYNAMICS SIMULATIONS

Deformation at nanoscale based on nanomechanics couples the principles of traditional mechanics and load applications with the fundamental aspects of chemistry and solid state physics. The movement of atoms in atomistic level systems can be analyzed through molecular dynamics that moves the atoms using classical mechanics equations of motion according to the inter-atomic force models from chemistry. Such equations of motion can be used to determine

the equilibrium (and minimum energy) structures or explore non-equilibrium dynamics. The atomistic scale dynamic deformation analyzes the dynamic moving locations of atoms via integration of the equations of motion. Computationally, the Newton's equations of motion applied at each atom are numerically integrated. The classical equation of motion is given by

$$F_i \equiv -\nabla_i \Pi(X) \equiv -\frac{\partial \Pi(X)}{\partial X_i} = m_i \frac{d^2 X_i}{dt^2} \equiv m_i \frac{dv_i}{dt} \equiv m_i a_i \quad (1)$$

where $\Pi(X)$ is the potential energy of the system, X_i and m_i are the atomic positions and masses of each atom, and t is the time. The terms v_i and a_i represent the velocity and acceleration of each atom. The potential energy Π depends on the atomistic material and is given by an analytical expression that yields energy as a function of the relative position of the atoms. Several potential energy functions exist for different materials and other multi-material systems. One such potential is Embedded Atom Method (EAM) potential and is used in the present study^{2, 3}.

Any molecular level system can be completely formulated by the positions X and velocity V (or momentum P) of atoms. The above equation is similar to the Newton equations of motion employed in continuum mechanics, but applied at the atomistic level. The dynamic behavior of the time-dependent atom motion is computed using an integrator such as the Verlet integrator⁴ to calculate the trajectories of the atoms. The time-scale involved in the MD simulations is of the order of $O(10^{-12} - 10^{-15} \text{ sec})$ and the length-scale is of order $O(10^{-10} - 10^{-7} \text{ m})$. The molecular dynamics simulator employed in the present study is LAMMPS (Large-Scale Atomic/Molecular Massively Parallel Simulator) from Sandia National Laboratory⁵.

TENSILE DEFORMATION OF NICKEL NANOWIRES

The nano scale tensile and flexural dynamic deformation behavior of the Nickel (Ni) nanowires due to tensile loading and flexural bending are presented in this section. The stress-strain constitutive behavior, tensile strength and the Young's modulus for various Ni nanowire configurations are presented and discussed. The natural frequency of the flexural deformation of these nanowires in a beam configuration via molecular dynamics simulations is obtained and analyzed. The simulation analysis of the deformation behavior in metallic nanowires modeled as atomic systems at finite temperatures is a dynamic process and is conducted using classical molecular dynamics.

Prior work in the literature exists on the deformation of the Copper and Gold nanowires⁶⁻¹² and carbon nanotubes¹³ via molecular dynamics (MD) methods. In the present paper, the tensile

deformation behavior of a nanowire configuration formed from the single crystals of nickel in the <001> (longitudinal), <100> and <010> (transverse) directions is considered. Most of the current literature is focused on the tensile deformation of nanowire configurations of different materials with very limited and non-existent work on the bending deformation behavior of nanowires. In addition to this tensile deformation, flexural deformation of the Nickel nanowires in a beam configuration is also presented.

Stress Definition in Nanoscale Deformation (Virial Stress)

A virial stress definition is used (Zhou, 2003) (Zimmerman et. al., 2003) to describe the macroscopic (continuum) stress in accordance with the microscopic/nanoscale, atomistic quantities^{14, 15}. Given the phase status of atoms, the macroscopic stress tensor in a macroscopically small, but microscopically large volume Ω is given by:

$$\sigma_{\alpha\beta} = \frac{1}{\Omega} \sum_{i \in \Omega} \left\{ -m_i (v_{i,\alpha} - \bar{v}_\alpha)(v_{i,\beta} - \bar{v}_\beta) + \frac{1}{2} \sum_j (X_{j,\alpha} - X_{i,\alpha}) f_{ij,\beta} \right\} \quad (2)$$

where

$$f_{ij,\beta} = \frac{\partial \Pi_i}{\partial X_j} \quad (3)$$

Here m_i is the mass of the i -th molecule in Ω , X_i is its position (α and β indicates Cartesian components), v_i its velocity, \bar{v} the local average velocity, and f_{ij} is the force on molecule i exerted by another molecule j . This virial stress is used to compute the stress values for the tensile and flexural deformation behavior in the present paper.

Computational Model Configuration of Tensile Nickel Nanowires

Figure 1 shows the computational model configuration of nickel nanowires employed. The nickel nanowires are made of Nickel FCC crystals with initial surface orientation of <100>, <010> and <001>. The lattice constant of faced-centered cubic (FCC) nickel crystal is $a=3.52$ Å¹⁶⁻¹⁸.

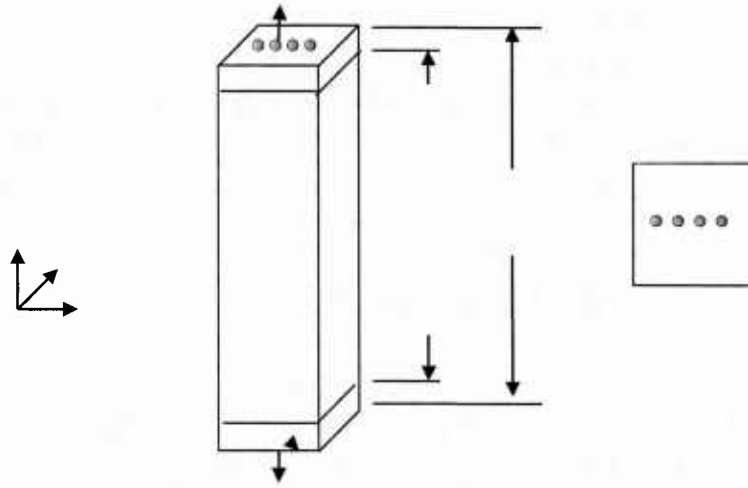


Fig. 1. Configuration of Tensile Nickel Nanowires for Tensile Deformation

The length (L) of Ni nanowires was taken to be 60λ in the $\langle 001 \rangle$ direction for the tensile deformation behavior. Different cross-sectional sizes that range from 5 to 20 λ for a side formed the cross-section of the nickel nanowire configurations. Constant velocities $\pm V_0$ are enforced on the top and bottom layers of the nanowires to emulate the tensile deformation. These top and bottom layers define the boundary layer and have identical size of 1 lattice constant, along the $\langle 001 \rangle$ crystalline direction.

Based on the velocities over boundary layers ($\pm V_0$), the atomistic nanowire model system deform with a strain rate given by:

$$\dot{\epsilon}' = \frac{2V_0}{L} \quad (4)$$

where L is the length of nanowires⁹.

Different velocities were employed in the simulations to give different strain rate conditions. Table 1 presents the velocity of boundary layers (V_0) and the resulting strain-rate. For the nanoscale system, the velocity is expressed in terms of lattice-constant (λ) per pico-second.

$\dot{\epsilon}(\text{1/s})$	$V_0 (\lambda/\text{ps})$
1.67×10^7	5×10^{-4}
1.67×10^8	5×10^{-3}
1.67×10^9	5×10^{-2}
1.67×10^{10}	5×10^{-1}

Table 1. Tension Velocity of Nanowires and the Resulting Strain Rate

Effect of Strain Rate

Figure 2 presents the tensile stress-strain behavior obtained for the two nickel nanowire configurations with dimensions of $5 \times 5 \times 60$ and $10 \times 10 \times 60$ at various strain rates ($1.67 \times 10^7 - 1.67 \times 10^{10} \text{ s}^{-1}$). These tensile deformation are obtained from MD simulations based on an NVE ensemble at temperature $T=300\text{K}$. It is noted that a higher strain-rate led to higher oscillations in the stress-strain curve due to the time dependent nature of the strain application and the associated dynamic stress.

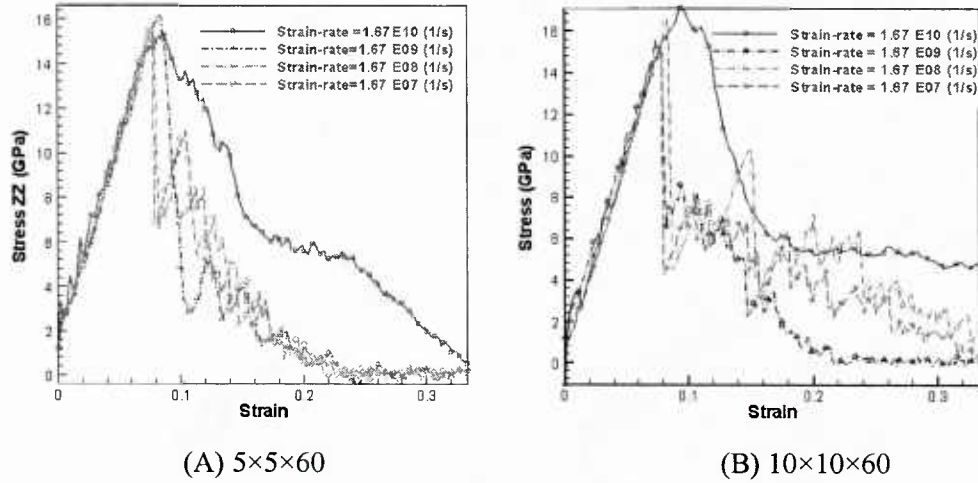


Fig. 2. Stress-Strain Curve of Nickel Nanowires under Various Strain-rates at $T=300\text{K}$.

The Young's modulus is determined from the tensile stress – strain curve for the strain $\varepsilon < 0.08$ using a linear regression. Table 2 and 3 shows the Young's modulus and maximum yielding stress of nickel nanowires respectively under tensile loading.

Strain-rate($\dot{\varepsilon}$)	$5 \times 5 \times 60$	$10 \times 10 \times 60$
1.67×10^7	191.27	184.2127
1.67×10^8	189.82	192.0556
1.67×10^9	184.33	190.3636
1.67×10^{10}	182.71	187.2973

Table 2. Young's Modulus (GPa) of Nanowires with Various Strain-rates

These results indicate that strain rate does not significantly influence the Young's modulus and the maximum yield stress.

Max. Yielding Stress (GPa)	5×5×60	10×10×60
1.67×10^7	15.7178	15.4835
1.67×10^8	15.6100	16.6046
1.67×10^9	16.2640	15.6100
1.67×10^{10}	15.3314	17.0836

Table 3: Maximum Yielding Stress (GPa) of Nanowires with Various Strain-rates

Figures 3 and 4 present the progressive deformation and failure of 5×5×60 Ni nanowires for the strain rates 1.67×10^7 and 1.67×10^{10} (s^{-1}) respectively. The deformation behavior indicates that the yielding slip planes, cross slip and the breaking neck^{6, 12, 18} of nickel nanowires are influenced by the strain-rate. The two deformation configurations presented in figures 3 and 4 are an intermediate configuration during the deformation and the final yielding configuration. These figures clearly show that the strain rate influences the yielding slip planes, cross slip, and the breaking neck during the tensile deformation of Ni nanowires.

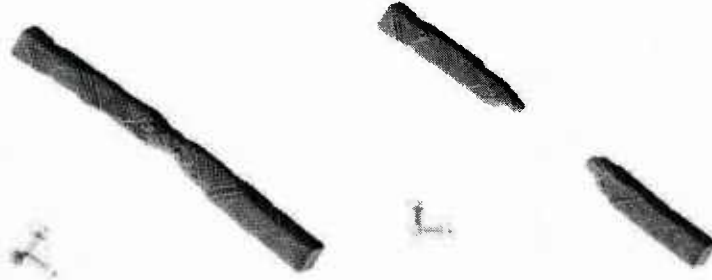


Fig. 3. Tensile deformation and failure of 5x5x60 nickel nanowire; strain rate = 1.67×10^7 (s^{-1})



Fig. 4. Tensile deformation and failure of 5×5×60 nickel nanowires; strain rate= 1.67×10^{10} (s^{-1})

FLEXURAL DEFORMATION OF NICKEL NANOWIRES

The flexural deformation behaviour of nickel nanowires due to flexural bending based on their atomistic configurations are discussed and presented next. In particular, the deformation

vibration frequencies obtained from the molecular dynamics simulations are compared with the natural frequencies based on classical beam theory under two different boundary conditions.

Computational Model Configuration and Analysis of Flexural Nickel Nanowires

Figure 5 presents the configuration of Nickel nanowire beams and the corresponding molecular model. The molecular model configuration is based on single crystals of Nickel in the $\langle 001 \rangle$ (longitudinal direction), $\langle 010 \rangle$ and $\langle 100 \rangle$ (transverse directions) directions and with a dimension of $120 \times 10 \times 1$ cubic lattice constants (rectangular cross section with a longer span). Two types of boundary conditions are considered in the flexural deformation: 1. Both ends pinned (i.e., simply supported), 2. Both ends clamped. The nanowire beam deflects under the action of applied loading and when the loading force is removed, the displaced beam would try to return to its original position. The inertia of the beam would cause the beam to vibrate. The transient flexural bending dynamic behavior of the molecular configuration of Nickel nanowire beams are investigated and analyzed.

The transient molecular dynamic simulations compute the new position of the atoms in the Nickel nanowire beam subjected to the flexural loading and the boundary constraints. The time increments are however significantly small in these dynamic simulations. A Mean Square Displacement (MSD ($u(t)$)) is defined and used as a measure of the average distance an atom in the model travels over a certain time interval period. This is defined as:

$$msd(u(t)) = \frac{1}{N} \sum_{i=1}^N u_i^2(t) = \frac{1}{N} \sum_{i=1}^N (r_i(t) - r_i(0))^2 \quad (5)$$

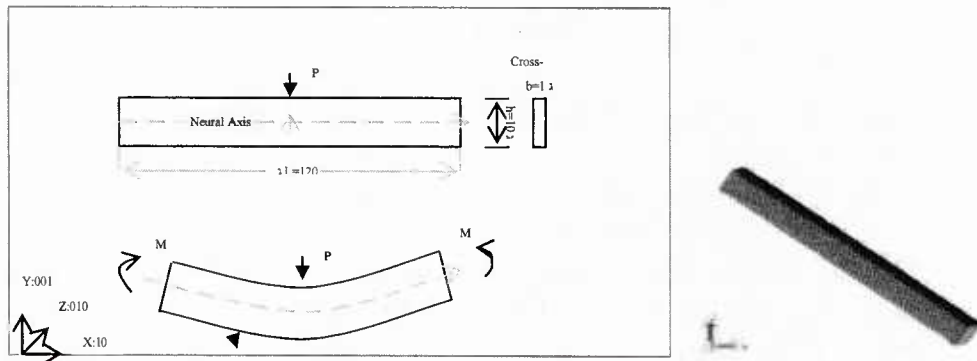


Fig. 5. Configuration of Nickel and Molecular Model of Nanowire Beams for Flexural Deformation

The displacement $\mathbf{u}_i(t) = \mathbf{r}_i(t) - \mathbf{r}_i(0)$ is the distance traveled by molecule i over some time interval t , and the squared magnitude of this vector is averaged over many such time intervals. This MSD displacement value is used in the analysis of time dependent displacement response of the Ni nanowire flexural beam configuration. The dynamic displacement responses under two different boundary conditions for flexural bending are presented next.

Simply Supported Nickel Nanowire Beam

The Nickel nanowire beam configuration as shown in figure 5 is simply supported (rotations are possible at the ends) and is subjected to a dynamic concentrated point load at the center of the nanowire beam. Two different load values ($F=0.01\text{eV}/\text{\AA}$ and $0.03\text{eV}/\text{\AA}$) are analyzed.

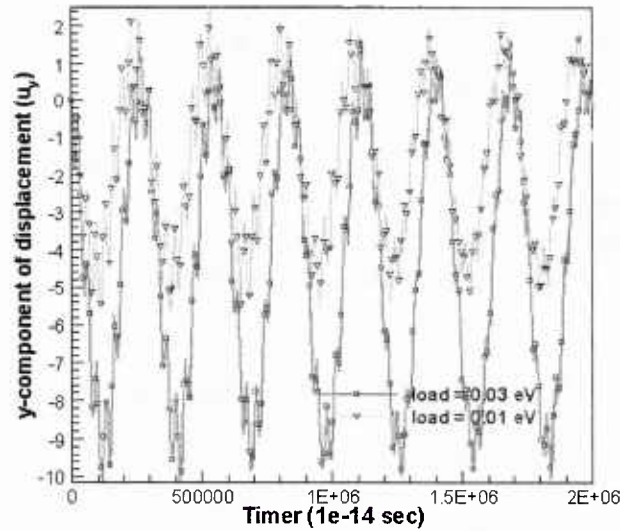


Fig. 6. Transient Dynamic Vertical Displacement at the Center (simply supported ends)

The dynamic vertical displacement at the center is proportional to the loading value and increases with a higher magnitude of external loading. The natural frequency of the dynamic vibration as computed from the above displacement - time profile is however independent of the magnitude of external loading. The computed angular frequency from the predicted time dependent deflection of the molecular model of the Nickel nanowire beam shown in figure 6 is $2.4166\text{E}+09$ Hz.

The natural frequency for the case of a simply supported beam based on classical beam theory analysis is given by¹⁹⁻²¹

$$\omega_n = (n\pi)^2 \sqrt{\frac{EI}{\rho AL^4}} \quad (6)$$

Using a Young's Modulus value of $E = 190 \text{ GPa}$ (1.1859 eV/\AA^3) obtained from the tensile stress strain deformation of Nickel nanowires discussed earlier, the mode 1 frequency value based on the classical beam theory is $2.5244E+10 \text{ Hz}$. This frequency as obtained from the classical beam theory is at least one order higher than the frequency obtained from the time dependent deflection using molecular dynamics simulations. The classical elastic beam theory based on continuum mechanics principles also indicate that the natural frequency of vibration of a simple supported beam is independent of the magnitude of the external loading and depends only on the beam cross sectional moment of inertia, cross-sectional area, length and modulus of elasticity of the material. The natural frequency obtained from molecular dynamics simulations for the loading and simply supported boundary conditions as presented in figure 6 is also independent of the magnitude of the external loading.

Clamped Nickel Nanowire Beam

The nanowire beam as discussed earlier is fixed at both ends (displacement and the rotation at the ends are zero) and is subjected to external loading force at the center. As before, two different loading values are investigated. Figure 7 presents the computed dynamic displacement response of the loaded center of the nanowire beam. As seen from figure 7, the dynamic displacement magnitude depends on the external loading value while the frequency of the dynamic displacement is independent of the external loading values. This is in direct correlation with the analytical results of natural frequency based on the classical beam theory.

The computed angular frequency obtained from the predicted time dependent deflection of the clamped Nickel molecular beam shown in Figure 12 is $2.3271E+09 \text{ Hz}$. The frequency of vibration based on the classical beam theory for this case of clamped ends is given by¹⁹⁻²¹

$$\omega_n = (K_1 L)^2 \sqrt{\frac{EI}{\rho AL^4}}; \quad K_1 = 4.73 \quad (7)$$

Using the same Young's Modulus for the Nickel nanowire as before, the mode 1 natural frequency as obtained based on the classical beam theory is $5.7225E+10 \text{ Hz}$. This frequency obtained from the classical beam theory is at least one order higher than the frequency obtained

from the time dependent center point load deflection using the deformation behavior from the molecular dynamics simulations. The classical beam theory based on continuum mechanics principles also indicate that the natural frequency of a clamped beam is independent of the external loading and depends only on the beam cross sectional moment of inertia, cross-sectional area, length, and modulus of elasticity of the material. This was also the case in the frequency of the nanowire beams obtained from molecular dynamics simulations as presented in figure 7.

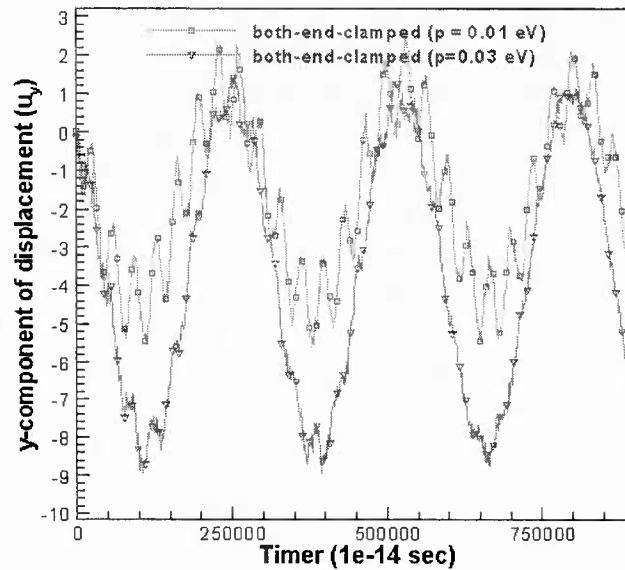


Fig. 7. Transient Dynamic Vertical Displacement at the Center (clamped ends)

DYNAMIC CRACK PROPAGATION IN NANOSCALE MATERIAL SYSTEMS

Fracture and failure of simple and complex materials remain a fundamental problem for engineering and research community. Metal/metal interfaces with mismatch in physical and mechanical properties are frequently encountered in a broad range of products of technological importance; examples include wear-resistant and fatigue-resistant coatings for nano-materials. In many of these applications catastrophic failure occur, when a crack initiated at the surface reach the interface between the surface material layer and the base material.

Dynamic fracture has been studied experimentally and by large-scale atomistic simulations in various materials²²⁻²⁷. Molecular Dynamics modeling of deformation due to fracture provide time-dependent behavior of a propagating crack. The crack propagation under mode I loading in a Ni single crystal and a Ni-Al bimetal interface system with a crack initiated and propagating from the Ni surface layer towards the Ni-Al bimetal interface are presented in the present paper.

Analysis Methodology

Molecular Dynamics modeling employing the embedded atom method (EAM) inter-atomic potential^{2, 28, 29} was used to investigate properties of a (001) [100] crack system under mode I loading in both the Ni and Ni-Al bi-metal interface systems. The EAM functions for Ni-Ni and Ni-Al interactions used are the ones that were recently developed by Pun, et al³⁰.

The material configuration employed for the MD simulations and analysis is a strip geometry as shown schematically in fig. 8 for both the Ni and Ni-Al system. Free boundary conditions were applied in the x and z directions and periodic boundary condition was applied in the y direction (with plane strain condition). The x, y and z axes are along the [100], [010] and [001] crystallographic directions, respectively. For the (001) [100] crack system, the crack-free surfaces are (001) and the crack propagates along the [100] direction. For Ni, the simulation slab has dimensions of $199a_{Ni} \times 7a_{Ni} \times 62a_{Ni}$ with 349,125 atoms, where a_{Ni} (3.52 \AA) is the lattice parameter of Ni. This molecular system configuration is believed to be large enough to take care of the long-range character of the crack strain fields.

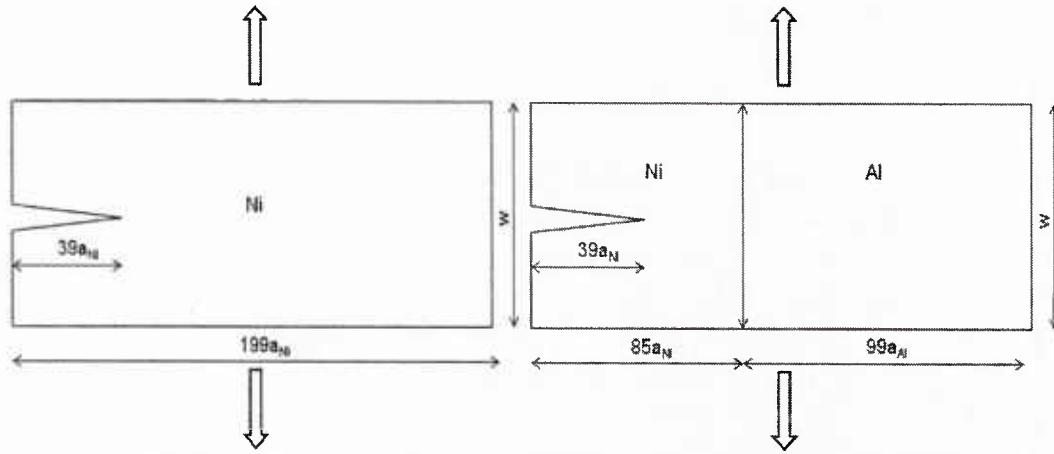


Fig 8. Schematic of Strip Geometry for Ni and Ni-Al Bimetallic Interface

The nanoscale Ni-Al bilayer material configuration was created and assembled from two semi-infinite perfect crystals of Ni and Al with an orientation relationship of $[100] \parallel [100]$, $[010] \parallel [010]$ and $[001] \parallel [001]$. The thermodynamic and geometric factors both equally play an important role in determining atomic structures of bi-metallic interfaces. The two dimensions in the y and z directions were therefore, not chosen arbitrarily (due to lattice size mismatch of Ni and Al) but determined such that the strains imposed on the Ni and Al semi-infinite perfect crystals is minimum and the periodic boundary condition is ensured in the y direction. The Ni crystal has 7 periodicity-lengths in the y direction and 62 periodicity lengths in the z direction, whereas Al crystal has 6 periodicity lengths in the y direction and 54 periodicity lengths in the z

direction. The lengths of Ni and Al crystals in the x-direction were chosen to be 29.92 nm and 40.095 nm, respectively. The total calculated dimensions of 70.015 x 2.464 x 21.87 nm in the three directions in Ni-Al were found to be comparable with the corresponding three dimensions of 70.048 x 2.464 x 21.824 nm in the Ni single crystal. The energy of the nanoscale bi-layer was first minimized using conjugate-gradient energy minimization technique. The system was then relaxed using MD in NPT ensemble to a pressure of 0 Bar and a temperature of 0°K. The relaxed semi-coherent structure showing atomic configuration at the Ni-Al interface, when viewed down the [100] (x) direction is shown in fig. 9. In the figure Al and Ni atoms are shown in silver and brown, respectively.

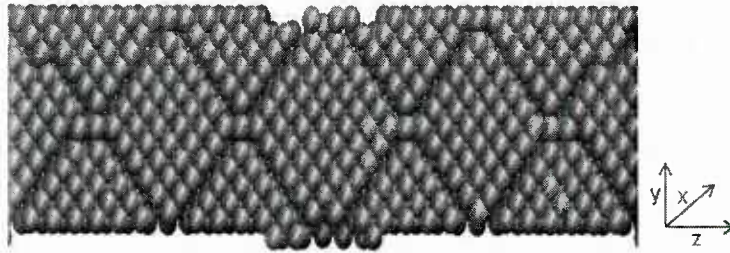


Fig. 9: Atomistic structure of the Ni-Al interface viewed along [100] (x) direction. Al and Ni are shown in silver and brown, respectively.

For both the Ni and Ni-Al an initial crack of length $39a_{Ni}$ (a_{Ni} being the lattice parameter of Ni) along the x-direction was introduced into the lattice by partially turning off inter-atomic interactions between atoms in eight consecutive (001) planes. The two middle planes constituted the upper and lower surfaces of the initial crack. The slab was initialized at zero temperature and an outward strain rate of $1 \times 10^9 \text{ sec}^{-1}$ was imposed on the outer most columns of atoms defining the upper free surface of the slab in the z direction. A linear velocity gradient was applied across the slab and that applied an increased outward strain with time in the z direction creating the Mode I fracture loading condition. This external loading leads to crack growth and propagation that can lead to eventual structural failure of the material. Molecular dynamics simulations presented in this work were conducted using molecular dynamics solver, LAMMPS⁵.

Deformation Due to Crack Propagation

The crack growth and propagation was studied on a (001) plane for both the Ni and Ni-Al nanoscale material system. The strain energy release rate (G) is the amount of energy per unit area that is supplied by the elastic energy stored in the system. This is calculated by integrating the stress-strain curve with respect to strain, ϵ . In the present molecular system configuration, this is given by

$$G = w \int_0^{\varepsilon} \sigma_z(\varepsilon') d\varepsilon' \quad (8)$$

where, w is the width of the strip in the z direction and σ_z is the z component of the stress. The calculated stress-strain curves for Ni, Ni-Al and Al are shown in fig. 10. The stress for each atom is due to its interaction with all other atoms in the system (within the force cut-off). Atomistic per atom stresses, given by a “stress times volume” formulation, as implemented in LAMMPS were calculated and summed over all the atoms of the system to get σ_z component of the stress. The sum was normalized by the system volume to finally compute σ_z . As expected, σ_z increases with strain to a certain value and then decreases for all the three systems. The maximum reached value of σ_z was found to be 7.56 GPa for Ni, 4.72 GPa for Ni-Al bimetal system, and 3.69 GPa for Al.

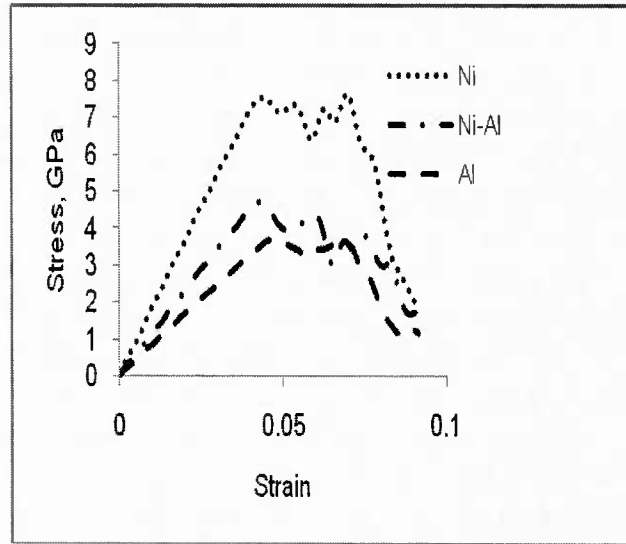


Fig.10: Stress-strain curves for Ni, Ni-Al and Al

According to Griffith's criteria, a brittle crack under mode I loading propagates when G corresponding to an applied load is equal or greater than $2\gamma_s$, where γ_s is the surface energy of each plane of the crack. From the stress-strain curve, the calculated critical strain energy release rate at which the crack starts to propagate in Ni (G_{cNi}) is 3.86 J/m² and in Ni-Al (G_{cNiAl}) is 2.4 J/m². The corresponding given values of the Griffith load from the EAM potential, which is

twice the (001) surface energy (γ_s), are 3.756 J/m² for Ni (G_{Ni}) and 1.886 J/m² for Al (G_{Al})³¹. The value for the critical strain energy release rate in Ni-Al (G_{cNiAl}) in the present bimetal system is between the Griffith loads for Ni and Al. The atomistic per atom stresses, as mentioned above, were summed over all the atoms of the system before normalizing the sum by the system volume to get σ_z component of the stress. In the Ni-Al bimetal interface system they were summed over both the Ni and Al atoms. The maximum attained stresses in the Ni, Ni-Al and Al systems as obtained from the simulations are in the order $\sigma_{zNi} > \sigma_{zNiAl} > \sigma_{zAl}$. The large difference in the maximum attained stresses (and therefore critical energy release rate) in Ni and Ni-Al is due to the inclusion of the per atom stresses from Al atoms in computation of σ_z for the Ni-Al system. The strain energy release rate, however, is related to the stress distribution around the crack tip rather than the stresses of the whole system.

Temperature control, which may affect crack-tip dynamics, is not applied in this work. While the initial temperature of the system is nearly zero, it increases as the crack advances. However, the average temperature of all the three systems remained below 50°K during the entire simulations. The snapshot pictures at various dynamic crack propagation simulation times showing mechanisms of crack propagation in Ni and Ni-Al are shown in figs. 11 and 12. In both of the figures the atoms are colored according to the centro-symmetry parameter³², which is a scalar quantity designed to identify defects such as interfaces, stacking faults and dislocations. In all of the images, atoms with a centro-symmetry parameter (P) close to zero were removed to facilitate easier viewing of the defects inside the structures. The visible atoms are associated with crack surfaces, exterior slab surfaces (only three surfaces are shown), Ni-Al bi-interfacial layer and other defects created during crack motion. The atoms are colored with yellow for dislocations (P 0.5 – 4.0), brown for stacking faults (P 4.0 – 12.0), and green for surface atoms (P > 12). The yellow and brown are also associated with atoms with crystallinity other than FCC. For both the Ni and Ni-Al, the early time sequence of the crack propagation at simulation times of 40, 45 and 50 ps (fig. 11) show that the crack initially moves in a straight line with ‘mirror’ cleaved surfaces. The crack surfaces in Ni (fig. 12) then began to roughen starting at around 55 ps with crack eventually ceasing to continue further with proliferation of dislocations at the crack tip at a simulation time of 65 ps. In Ni-Al (fig. 12) as crack nears interface a bud at the crack tip called ‘process zone’ began to grow at 55 ps along with roughening of the surfaces of the crack. The process zone that represents a local disorder at the crack tip shows no apparent plasticity during the initial brittle cleavage of the surfaces. At 65 ps, when the propagating crack tip lie roughly at 8.5 nickel lattice spacing (8.5 a_{Ni}) from the interface, the dislocations start emanating from the interfacial bi-layer and they start traveling away from the interface towards the bulk Al. When compared to Ni the crack tip in Ni-Al at 65 ps lags behind the crack tip in Ni by about 15.5 nickel lattice spacing (15.5 a_{Ni}).

The snapshot pictures showing an enlarged and a close-up view of the defect structures formed at the crack tip after initiation of plastic deformation at 65 and 70 ps in Ni are shown in fig. 13. The atoms are colored as described above with yellow for dislocations, brown for stacking faults, and green for surface atoms. The snapshots at 65 and 70 ps show formation and evolution of stacking faults associated with nucleation of dislocations from the crack tip. The stacking faults are bounded by dislocation loops, which start at the crack tip. The appearance of dislocations at the crack tip suggests a dynamic brittle-to-ductile transition which leads to a crack arrest in the Ni. When the surfaces of the crack began to roughen atomically, the crack attains a velocity of approximately one third of the Rayleigh wave speed. A plot of crack tip position versus simulation time for Ni is given in fig. 15. The initiation of the crack propagation is taken as the zero simulation time. The crack tip position is determined by comparing the relative distance between atoms of the upper and lower planes of the crack with a value for which the bond is believed to be broken. The slope of the linear part of the crack tip position versus simulation time gives velocity of the crack propagation (940 m/sec) to be about one third of the Rayleigh wave speed (using Rayleigh speed of 2797 m/s³³). This crack surface roughening has been identified as the onset of an intrinsic dynamical instability of the brittle fracture process by previous investigators^{26, 27}.

In Ni-Al bimetal system as discussed above, the crack surfaces initially grow brittle with crack surfaces getting roughened at around one-third of the Rayleigh wave speed. However, two regimes of crack propagation velocities were observed in this case (fig. 16). The first (960 m/sec) corresponds to one-third of the Rayleigh wave speed with which the crack starts to propagate after an initial transient time; the second (350 m/sec) regime corresponds with crack growth getting decelerated as it nears the interface after the onset of crack surface roughening and growth of the process zone at the crack tip. The snapshots showing structural evolution with time at 67, 68, 69 and 70 ps as the growing crack approaches the interface are shown in fig. 14. The atoms are again colored by centro-symmetry parameter with yellow for dislocations (P 0.5 – 4.0), brown for stacking faults (P 4.0 – 12.0), and green for surface atoms (P > 12). As the crack growth approaches the bi-metal interface, dislocations start emanating from the interfacial bi-layer and they start traveling away from the interface towards the bulk Al. At 68 ps, the ‘process zone’ at the crack tip start interacting with defects at the interface that eventually blunts the crack tip and ceases further crack growth ultimately prohibiting crack from propagating beyond the Ni-Al interface. However, the system continues to dissipate elastic energy through continued creation and motion of dislocations in Al. The snapshots in fig. 14 also show formation and evolution of stacking faults associated with nucleation of dislocations from the interfacial bi-layer. The stacking faults, which in this case start at the interfacial layer, are bounded by the dislocation loops (colored in yellow in fig. 14).

Summary and Concluding Remarks

The extreme smaller lengths at the nano level require analysis methodologies based on three-dimensional atomistic deformation characteristics for nanoscale material systems. In the field nanomechanics, principles of mechanics are employed in conjunction with interatomic potentials capturing the molecular forces, atomistic level interactions, and offer a potential to understand the associated deformation behavior at nanoscale. The present paper presented the tensile and flexural behavior of Nickel nanowires based on the dynamical behavior of their atomistic structures. The results clearly elucidate the applicability of the nanomechanics based atomistic modeling for the understanding of the behavior of such nanowires under mechanical loading conditions.

The atomistic modeling simulations and configurations were also employed to understand the dynamic crack propagation behavior in a nanoscale Nickel single crystal and a Nickel-Aluminum nanoscale bimetal interface. The embedded atom method interatomic potential are used to investigate the behavior of (001) [100] crack system under mode I loading. The dynamic crack propagation for Ni show an initial brittle crack propagation followed by a roughening of the crack surfaces at one-third of the Rayleigh wave speed. In Ni-Al, the crack surfaces initially grow brittle. Two regimes of crack propagation velocities were observed in this case with crack getting decelerated as it nears the interface. Further dynamic analysis of the crack propagation indicated a cease in the crack propagation in Ni due to a brittle to ductile transition. In Ni-Al bimetal interface system, as the crack approaches the interface, a process zone representing local disorder at the crack tip was observed to start growing and interacting with interfacial defects that eventually results in a blunting of the crack tip.

The fundamental understanding of nanoscale crack propagation evolving into multi-scale nano to continuum analysis are essential to optimize and ensure the safety and reliability of engineered structures with nanocoatings, when a crack initiated at the surface reaches the interface between the surface nanocoating material layer and the base material.

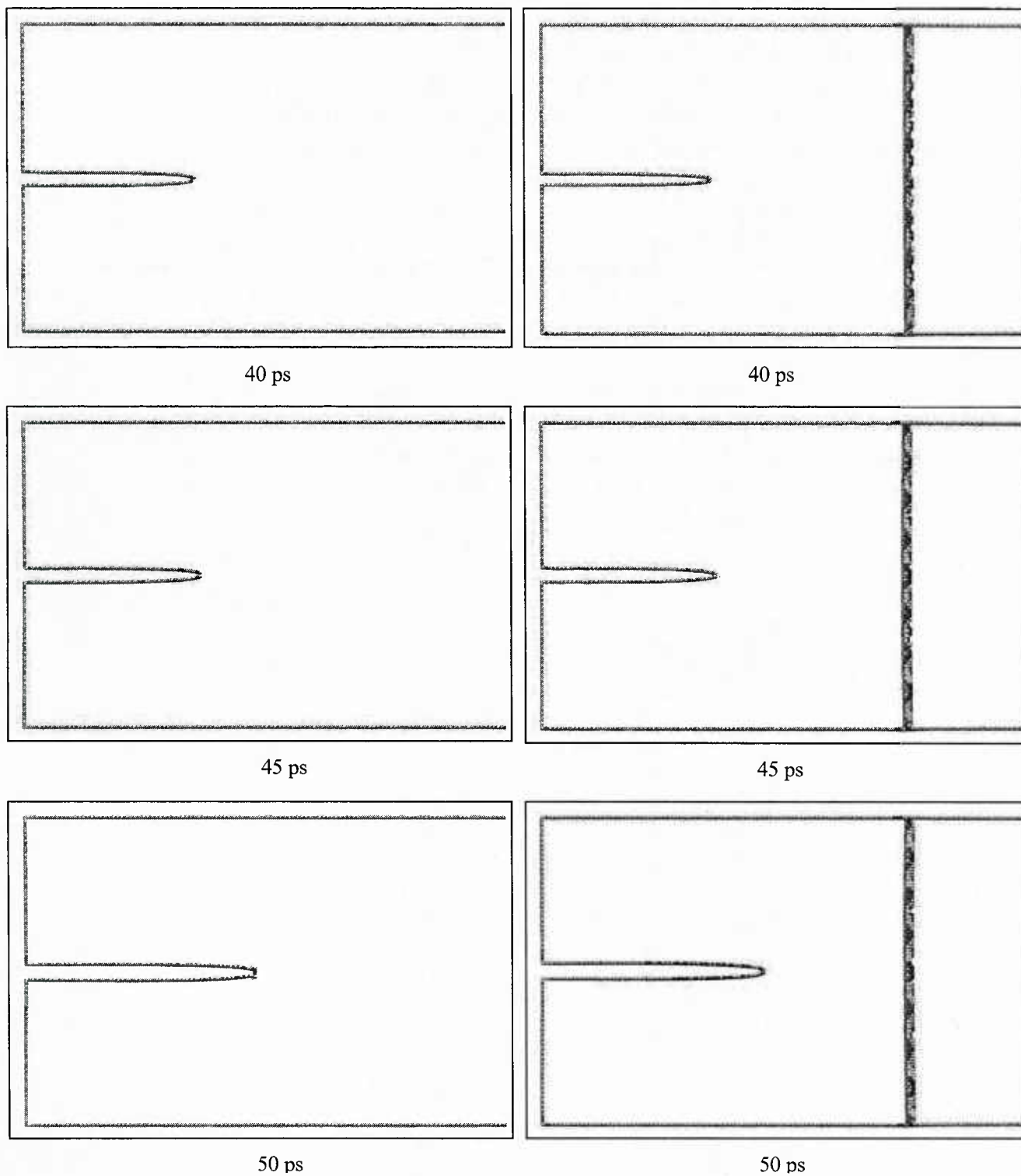


Fig.11: Simulation snapshots showing early-time sequence of crack propagation in Ni (left) and Ni-Al (right) at 40, 45 and 50 ps. Atoms are colored by centro-symmetry parameter.

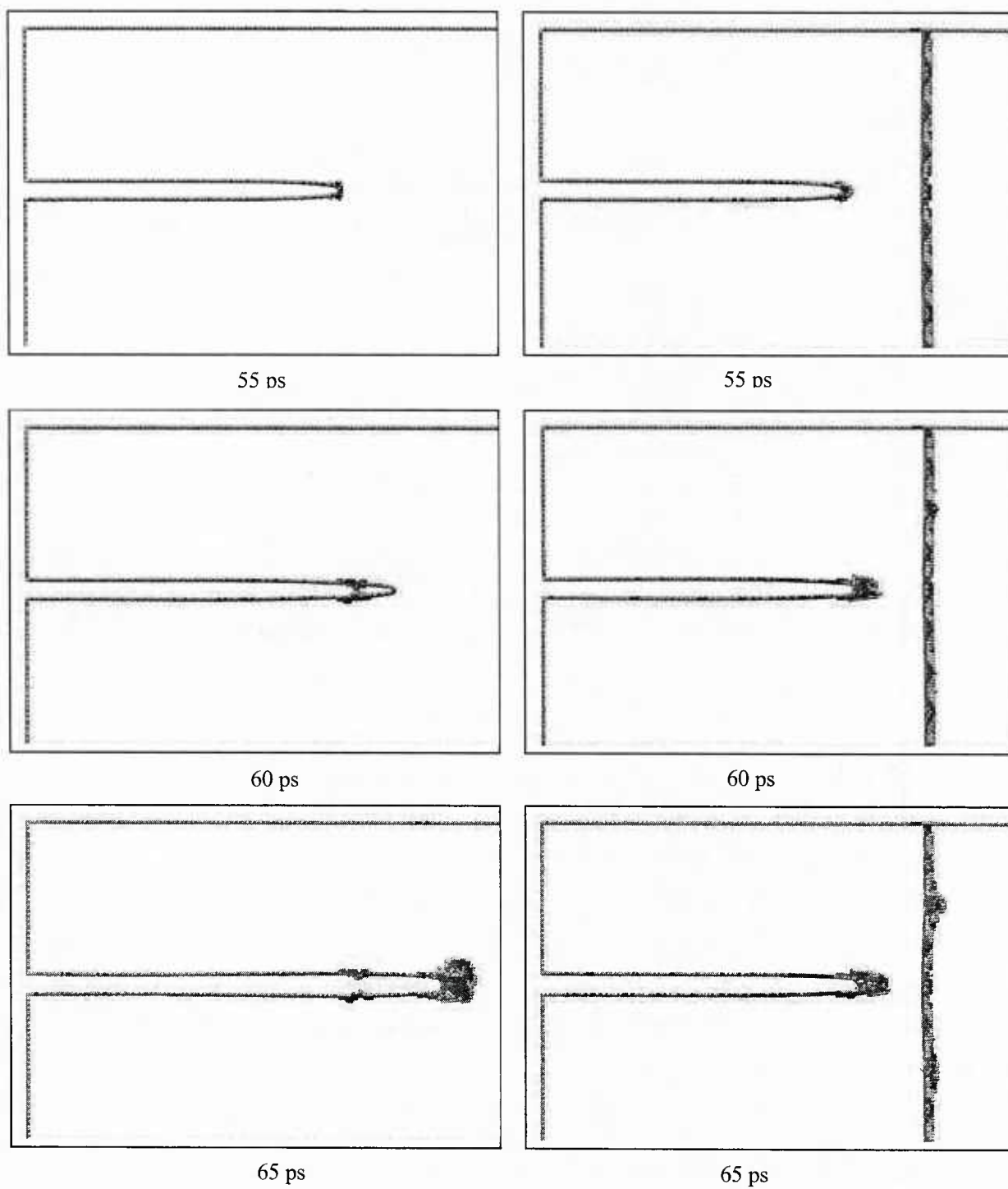


Fig. 12: Simulation snapshots showing late-time sequence of crack propagation in Ni (left) and Ni-Al (right) at 55, 60 and 65 ps. Atoms are colored by centro-symmetry

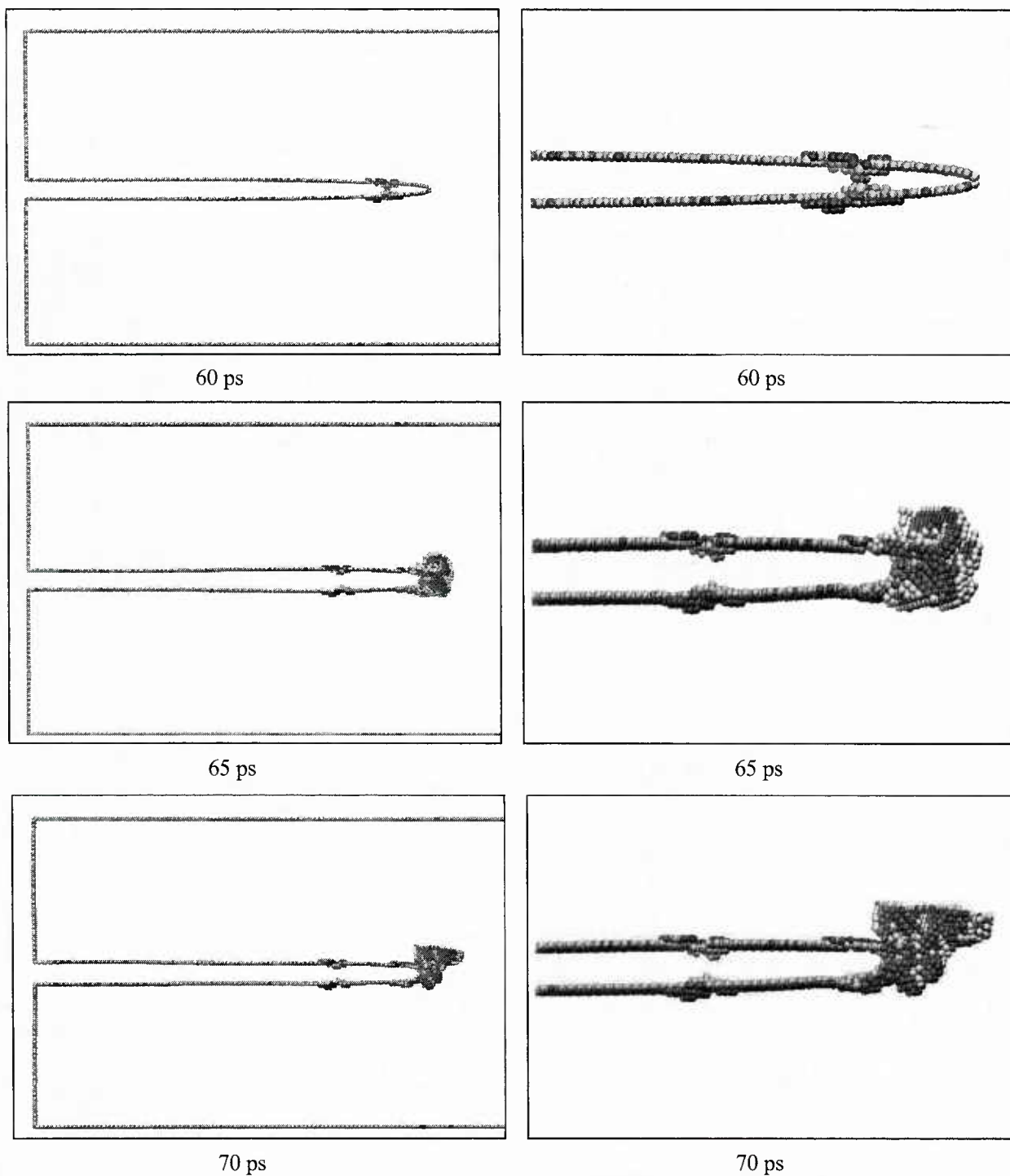
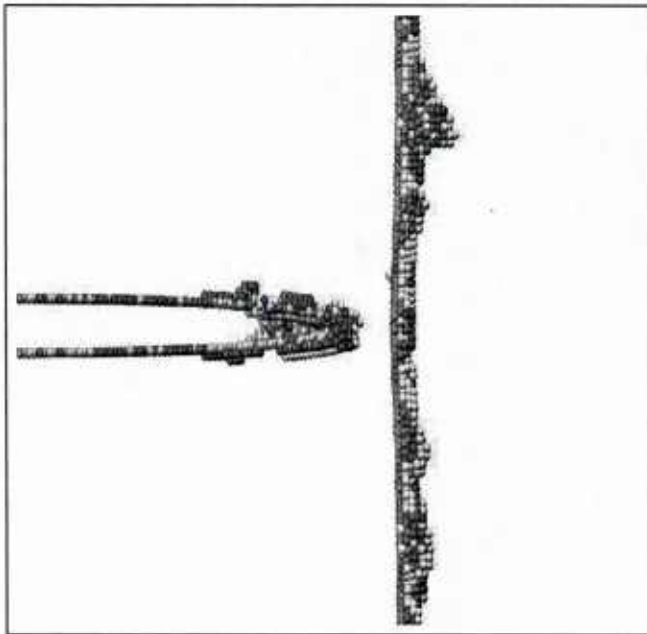
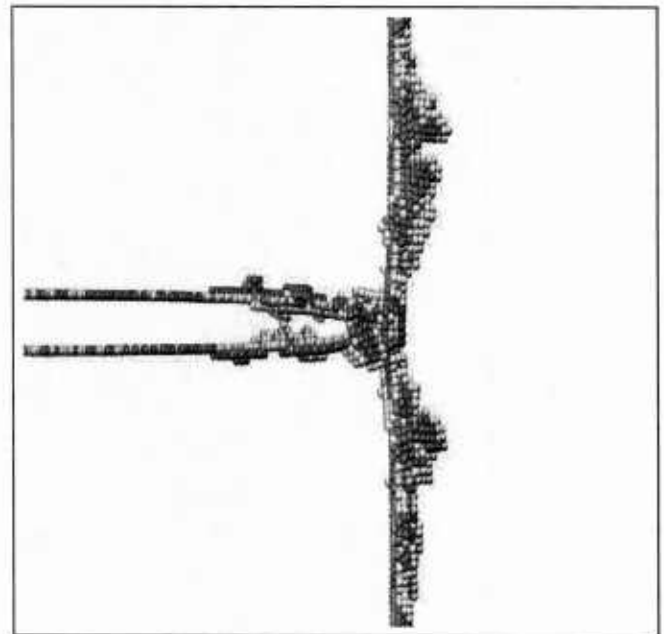


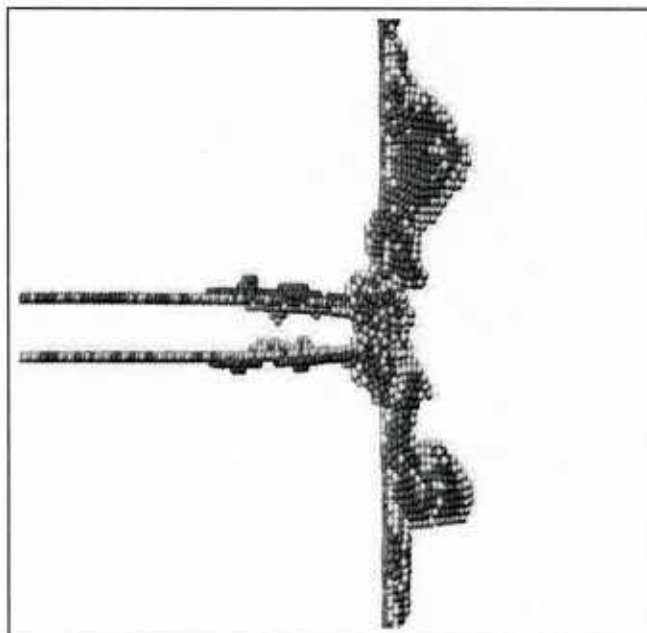
Fig. 13: Simulation snapshots showing crack propagation in Ni at 60, 65 and 70 ps. Left panel show three exterior slab faces. Right panel show a close-up view of the defect structures. Atoms are colored by centro-symmetry parameter with yellow for dislocations, brown for stacking faults and green for surface atoms.



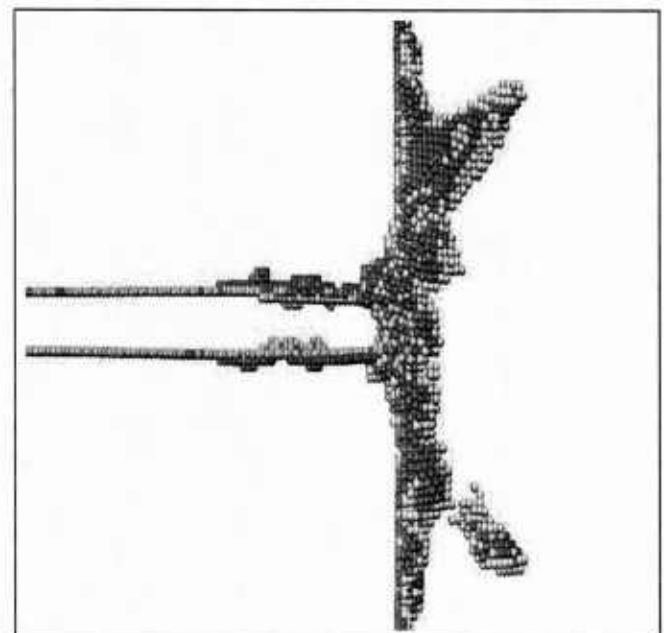
67 ps



68 ps



69 ps



70 ps

Fig. 14: Simulation snapshots showing crack propagation in Ni-Al at 67, 68, 69 and 70 ps. Atoms are colored by centro-symmetry parameter with yellow for dislocations, brown for stacking faults and green for surface atoms.

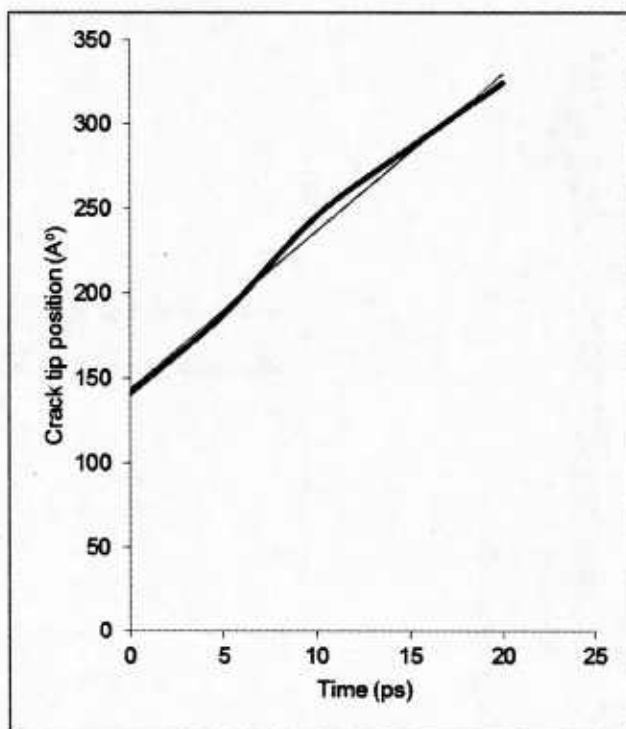


Fig. 15: Crack tip position versus simulation time for Ni.

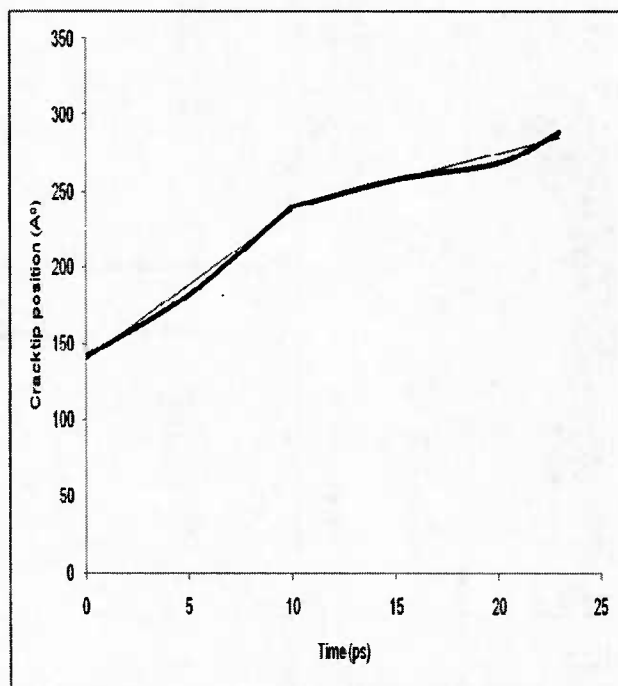


Fig. 16: Crack tip position versus simulation time for Ni-Al.

References

1. H. Chen, C. Lin, W. Chen, Y. Hsu and R. Uang, Proceedings of International Conference on Microsystems, Packaging and Assembly, Taiwan IEEE Paper 1-4244-073504/06, (2006)
2. S. M. Folies, M. I. Baskes and M. S. Daw, Physical Review B 33, 7983 (1986)
3. M. S. Daw and M. I. Baskes, Physical Review B 29, 6443 (1984)
4. K. S. Cheung and S. Yip, Modeling Simul. Mater. Sci. Eng. 2, 865 (1994)
5. S. J. Plimpton, Journal of Computational Physics 117, 1 (1995)
6. W. J. Chang and T. H. Fang, Journal of Physics and Chemistry of Solids 64, 1279 (2003)
7. H. Ikeda, Y. Qi, Y. Cagin, K. Samwer, W. L. Johnson and W. A. Goodman III, Physical Review Letters 82, 2900 (1999)
8. S. P. Ju, J. S. Lin and W. J. Lee, Nanotechnology 15, 1221 (2004)
9. W. W. Liang and M. Zhou, Size and Strain Rate Effects in Tensile Deformation of Cu Nanowires. In 44th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conference, Norfolk, VA, (2003),
10. H. S. Park and J. A. Zimmerman, Physical Review B 72, 54 (2005)
11. E. Z. Silva, A. J. R. Silva and A. Fazzio, Comp. Mater. Sci. 30, 73 (2004)
12. E. Z. Silva, Physical Review B 69, (2004)
13. K. M. Liew, X. Q. He and C. H. Wang, Acta Materialia 52, 2521 (2004)
14. J. A. Zimmerman, E. B. Webb, J. J. Hoyt, R. E. Jones and P. E. Klein, Modeling Simul. Mater. Sci. Eng. 12, (2003)
15. M. Zhou, A New Look at the Atomic Level Virial Stress: On Continuum Molecular System Equivalence. In Proc. Royal Soc. London A, London, (2003), pp 2347
16. D. Frankel and B. Smit, Understanding Molecular Simulations: From Algorithms to Applications. Academic Press, San Diego, (2001),
17. J. Haile, Molecular Dynamics Simulations: Elementary Methods. John Wiley & Sons, New York, (1997),
18. T. Schlick, Molecular Modeling and Simulation - An Interdisciplinary Guide. Springer-Verlag, New York, (2002),
19. T. H. G. Megson, Structural Stress Analysis. John Wiley & Sons, New York, (1996),
20. J. W. Tedesco, W. G. McDougal and C. A. Ross, Structural Dynamics - Theory and Applications. Prentice-Hall, (1999),
21. E. Volterra and E. C. Zackmanaglou, Dynamics of Vibrations. Charles E. Merrill Books Inc., Columbus, (1965), Vol. ASIN B000BSN9QS, p ASIN B000BSN9QS
22. J. Fineberg, S. P. Gross, M. Marder and H. L. Swinney, Phy. Rev. Lett. 67, 457 (1991)

23. J. Fineberg, S. P. Gross, M. Marder and H. L. Swinney, *Physical Review B* 45, 5146 (1992)
24. S. P. Gross, J. Fineberg, M. Marder, W. D. McCormick and H. L. Swinney, *Phy. Rev. Lett.* 71, 3162 (1993)
25. P. Gumbsch, S. J. Zhou and B. L. Holian, *Physical Review B* 55, 3445 (1997)
26. F. F. Abraham, D. Brodbeck, R. A. Rafey and W. E. Rudge, *Phy. Rev. Lett.* 73, 272 (1994)
27. F. F. Abraham, D. Schneider, B. Land, D. Lifka, J. Skorvia, J. Gerner and M. Rosenkrantz, *Materials Research Society Symposium Proceedings* 463, 187 (1997)
28. M. S. Daw and M. I. Baskes, *Physical Review Letters* 50, 1285 (1983)
29. M. S. Daw and M. I. Baskes, *Physical Review B* 29, 6443 (1984)
30. G. P. Pujara Pun and Y. Mishin, *Philosophical Magazine* 89, 3245 (2009)
31. Y. Mishin, D. Farkas, M. J. Mehl and D. A. Popaconstantopoluos, *Physical Review B* 59, 3393 (1999)
32. C. L. Kelchner and S. J. Plimpton, *Physical Review B* 58, 11085 (1998)
33. M. Karimi, T. Roarty and T. Kaplan, *Modeling Simul. Mater. Sci. Eng.* 14, 1409 (2006)

A-2-2 Mechanical Behavior of Nanoscale Metallic Composites – Dynamic Crack Propagation in Ni-Al Bilayer Composite

Authors: R. Mohan, Y. Purohit, A. Kelkar, North Carolina A&T State University

Published Journal Article: *Journal of Computational and Theoretical Nanoscience*, Vol. 12, Pages 1-10, 2015

ABSTRACT

Nanoscale multilayer metallic composites (NMMCs) contain significantly high volume fraction of interfaces and exhibit strengths much higher than that of bulk materials composing the structures. This strengthening has been attributed to the presence of interfaces between materials that differ in properties such as elastic modulus, lattice parameters, slip plane orientations and act as barriers to propagating dislocations. This paper presents a review of two major factors that influence the properties and behavior of the NMMCs: Interface structure, Strengthening/Deformation mechanisms. The influence of semi-coherent Ni (nickel) - Al (aluminum) interface on Mode-I crack propagation in nanoscale Ni-Al bilayer composite under tensile and cyclic loading conditions analyzed through computational modeling is discussed. Results for nanoscale Ni-Al bilayer composite showed initial brittle crack propagation with planar cleavage of atoms followed by crack surfaces getting roughened when crack propagation speed is about one-third of Rayleigh wave speed. In case of Mode-I tensile cyclic loading, crack was found to propagate either by fatigue cleavage of the atoms or by void nucleation in the regions near the crack tip, depending on the value of maximum strain applied. In Ni-Al bilayer composite studied, as crack approached the interface, dislocations start emanating from the interfacial layer. The creation of voids was found to slow down crack growth in both the Ni and Ni-Al at higher maximum applied strain during cyclic loading. Plastic deformation was found to dominate crack propagation during tensile loading that resulted in a slower crack growth than cyclic loading. In all cases, presence of semi-coherent interface in the nanoscale Ni-Al bilayer composite was found to prohibit crack from propagating beyond the interface.

KEYWORDS: Nanoscale metallic multilayers, interfaces, molecular dynamics, bimetallic nanolayer, crack propagation.

INTRODUCTION

Nanoscale multilayer metallic composites (NMMCs) consisting of alternating nanometer (< 100 nm) thick layers of two or more materials on a suitable substrate have been of keen interest to the materials community. Their potential for unique and technologically important combinations of properties that emerge as the individual layer thickness is reduced to the nanometer-scale make

them uniquely multifunctional materials. NMMCs exhibit high strengths, which at room temperature can approach one-half or one-third of the estimated theoretical strengths of constituents¹⁻⁵, improved ductility^{6,7}, and good thermo-mechanical stability of interfaces^{6,8,9}. In addition to mechanical properties, novel electronic, magnetic and optical behaviors that result from nanolayering, make NMMCs attractive for applications such as hard/wear resistant coatings, diffusion barrier coatings, x-ray optical elements, magnetic recording media and heads, and micro and nanotechnological devices/systems (MEMS and NEMS)¹⁰⁻¹⁵. Besides coatings on substrates, NMMCs also find applications as self-supported high strength foils for a variety of structural applications. Multilayer technologies can also have a profound impact on manufacturing processes by decreasing the amount of machining necessary between raw materials and the finished products.

The mechanical properties and behavior of metallic multilayers has been the subject of extensive research activity in the past decade. Nanoscale multilayer metallic composites contain extremely high densities of interfaces, and achieve very high strength levels. The high densities of interfaces at nanoscale are a contributing factor to these very high strength levels. Interfaces between dissimilar material layers that differ in properties such as elastic modulus, lattice parameter, defect energies, and slip plane orientations play a crucial role in determining the material strength at nanoscale. The dissimilarities in the properties between the material layers act as a strong barrier to slip transmission¹⁶⁻²⁴. In addition, as the layer thicknesses are reduced from micrometer to the nanometer scale, the strengthening mechanisms transition from the Hall-Petch model of dislocation pileups at the interface to the Orowan model of single dislocation bowing between layers, and finally to the interface crossing mechanisms²¹. The increased strength achieved in nanoscale multilayers can be attributed to the resistance of the interface to the transmission of a single glide dislocation. This single glide dislocation is considered to be a critical unit process at layer thickness less than 5 nm, and largely determines the maximum strength achieved in nanoscale multilayers. The maximum strength is dependent upon the atomic structures and properties of the interface. These differences in the atomic structure and properties of the associated nanoscale material layers also lead to different types of interfaces between the material layers and their associated strengthening mechanisms.

Theory and multi-scale modeling analysis^{17,20,22,16,25-31} (atomistic modeling, elasticity-based dislocation theory, dislocation dynamics simulations and crystal plasticity modeling) along with experiments have been used to elucidate a variety of novel aspects of the strength, plasticity and deformation of NMMCs. The deformation and behavior of NMMCs are influenced by the crystallographic structures and properties of the interfaces that influence the strength; layer thicknesses and the associated strengthening and deformation mechanisms at different length scale thicknesses. All these factors influence the properties and deformation behavior of

NMMCs, including their tensile and fatigue properties, response to large plastic deformation, and thermal stability.

The discussions present a review of the major factors that influence the properties and behavior of the NMMCs: Interface structure, Strengthening/Deformation mechanisms. The interface structure between two dissimilar metallic materials in NMMCs influence the deformation behavior under Mode-I crack propagation. Mode-I crack propagation in a semi-coherent metallic Ni (nickel)-Al (aluminum) bilayer under tensile and cyclic loading conditions analyzed through computational atomistic Molecular Dynamics (MD) modeling is discussed. The discussions in this paper are organized as follows. Major factors that influence the properties and behavior of NMMCs, in particular, Interface structure, Strengthening/Deformation mechanisms are presented in section 2 and 3. This is followed by brief discussions on prior literature on the atomistic level modeling to study the nanoscale metallic structures/interfaces in section 4 followed by discussions on the analysis of dynamic crack propagation in nanoscale Ni-Al bilayer composite.

INTERFACE STRUCTURE

The mechanical properties and deformation mechanisms of NMMCs depend strongly on the type of metallic materials constituting the multilayers, and on the type of interfaces that form between the two material layers³². Interface act as barriers to propagating dislocations and cause strain hardening of the nanoscale multilayer materials. Depending on the materials involved in the multilayers, interfaces in NMMCs can generally be classified into the following four categories: coherent, semi-coherent; incoherent and hybrid interfaces.

Coherent and Semi-Coherent Interfaces

Coherent interfaces form when the two metals have the same type of lattice structure (e.g., both face-centered cubic (fcc)) and the difference in the lattice parameters is relatively small, in the order of a few percentages. e.g., the interface between fcc/fcc Cu-Ni system with cube-on-cube orientation relationship^{17-19 23 16 33}. In such systems, the two layers are constrained so that no misfit dislocations can form to relax the stresses due to lattice mismatch, resulting in the development of high stresses along the interface (coherency).

Semi-coherent interfaces form between metals with the same lattice type but larger mismatch in the lattice parameters. Such interfaces are characterized by a network of misfit dislocations that are needed in order to accommodate the large lattice mismatch at the interface e.g., the interface between the fcc/fcc Cu-Ag system with cube-on-cube orientation¹⁷. In this case misfit dislocations relax the long-range coherency stresses and the interface between the misfit dislocations remains coherent.

In most multilayers comprised of metals with the same lattice type and a small mismatch however, both coherent and semi-coherent interfaces may form depending on the thickness of the individual layers³⁴. For relatively thin layers, fully coherent interfaces are more favored energetically; however, a loss of coherency usually occurs when the layer thickness exceeds some critical value. In the case of coherent and semi-coherent interfaces, where both materials have the same crystal structures, slip planes and directions are nearly continuous across the interface and such interfaces are therefore labeled transparent.

Strengthening in nanolayered composites with coherent interfaces is usually attributed to forces on glissile dislocations at or near interfaces caused by lattice mismatch (coherency), elastic mismatch (Koehler) and changes in core structure on passing from one layer to the other (chemical). However, research suggest that for materials with coherent interface, the most important effect on its strength is derived from the coherency strains^{17 16}. Earlier atomistic simulations performed by Hoagland et al.^{17 20}, showed that the peak strength in coherent Cu-Ni multilayers at layer thickness below 5 nm may be interpreted in terms of the high coherency stresses that must be overcome for single dislocation transmission.

In multilayers with semi-coherent interfaces, pre-existing networks of misfit dislocations often dominate plasticity¹⁷. The semi-coherent interfaces act as barriers to slip because of the residual coherency stresses in areas between the misfit dislocations. Other factors acting as barriers to slip include, the interaction between misfit dislocations and glide dislocations, and the creation of a step when crossing occurs. The core structures of misfit dislocations play an important role in affecting the way that misfit dislocations interact with the glide dislocations. Misfit dislocations may be very narrow in the plane of the interfaces, as they are in Cu-Ni, or wide, as in Cu-Ag. The wide core, closely spaced misfit dislocations in the latter case (Cu-Ag) effectively remove local coherency stresses, promote dislocation mobility and lead to weak interfaces³⁵.

Incoherent Interfaces

Incoherent interfaces form between materials with different lattice structures, where the slip planes and slip directions are discontinuous across the interface leading to negligible coherency stresses in the system. A typical example of the incoherent interface is an interface between fcc/bcc Cu-Nb system with Kurdjumov–Sachs (KS) orientation relationship³⁶, where interface form along the close packed planes of Cu and Nb (111) and (110), respectively and the Cu and Nb layers are oriented with respect to each other such that a $\langle 110 \rangle$ direction of Cu is parallel to a $\langle 111 \rangle$ direction of Nb in the interface plane (the interface plane is Cu//Nb^{37, 38} and within the interface plane, $\langle 110 \rangle$ Cu// $\langle 111 \rangle$ Nb). Atomic relaxations in the interface lead to local patches of high and low atomic coordination and periodic arrays of defects. Although periodic structures might occur, they do not sustain the large stresses that can develop in both the coherent and semicoherent interface systems (where two metals have the same crystal structure and the slip

planes and directions are continuous across the interface). Due to discontinuity of the slip planes and directions, incoherent interfaces are also called opaque. The crystallographic discontinuity of slip systems becomes a major factor that may inhibit slip transmission across the interfaces in these systems.

Computational modeling simulations using embedded-atom potentials have been used to study atomic structures of Cu-Nb incoherent interfaces^{28 29 26}. For the KS orientation relationship, atomistic simulations have identified a variety of possible atomic structures of Cu-Nb interface with nearly same formation energies. The atomic structure, referred to as KS1, is formed by directly combining the two semi-infinite perfect crystals of Cu and Nb according to the KS orientation relationship^{29 26}. The atomic structure, referred to as KS2, is formed by inserting a strained monolayer of Cu³⁷ as an intermediate layer between the adjoining crystals in the KS1 interface. The inserted monolayer, a perfect Cu³⁷ plane, is strained in a way so as to remove the patches of under coordination present in the KS1 interface^{29 26}, thereby stabilizing the interface configuration even though it contains a strained Cu monolayer. Insight gained from the analysis of the KS2 interface structure has been applied to predicting other pairs of materials that may also form interfaces that lead to improved radiation damage resistance, such as those observed in Cu-Nb multilayer thin-film composites.

The shear resistance and sliding mechanism of interfaces between Cu-Nb layered composites, as a function of applied in-plane shear direction and different interface atomic structures have also been studied using atomistic simulations²⁹. These simulation results indicate that the shear strengths of Cu-Nb interfaces are significantly lower than the theoretical estimates of shear strengths for perfect crystals, strongly anisotropic, spatially non-uniform, and strongly dependent on the atomic structures of interfaces. The mechanism of interface sliding involves glide of the interfacial dislocation loops that nucleate from the weakest regions of the interface.

The low shear strength of the interface and the large in-plane anisotropy of shear strength have significant implications for the interactions of glide dislocations, from either copper or niobium crystal, with the interfaces. In addition to the geometric factor of slip discontinuity in Cu-Nb layered composites, atomistic simulations reveal several important factors, directly related to the weak interfaces that hindered transmission of dislocations across the interfaces²⁸. The stress field of a glide dislocation approaching the interface exerts enough stress to locally shear the weak interfaces, resulting in its (dislocation's) absorption and spreading of its core in the interface plane, thereby hindering its transmission.

DEFORMATION/STRENGTHENING MECHANISMS

Experimental observations of high yield strengths in nanoscale layered materials cannot be explained by a simple extrapolation of scaling laws such as the Hall–Petch relationship. Deformation mechanisms in these materials depend on layer thickness, λ . When the layer thickness is on the order of several tens to several hundreds of nanometers (> 50 nm), the Hall–Petch effect is considered to be the main reason for the material’s strength increase^{1 39–43}. However, when the layer thickness becomes less (< 50 nm) than the distance required for dislocations to interact and form substructures in bulk materials, strength does not obey the Hall–Petch relation. The Hall–Petch model is based on dislocation pile-up mechanisms. Creation of dislocation pile-ups, at $\lambda < 50$ nm becomes difficult and essentially new mechanisms that are based on interactions of single dislocations and interfaces come into play. Interfaces becomes the controlling parameter of plasticity in NMMCs^{4 17–20 22 16 44 45}.

The flow strength as a function of the thickness of the individual layers, λ , for Cu–Cr, Cu–Nb, Cu–Ag, Cu–Ni, and Cu–304SS bi-material systems is shown in figure 1²⁰. The data fit a Hall–Petch relation ($\sigma \sim \lambda^{-1/2}$) between strength and layer thickness for $\lambda > 50$ nm. The Hall–Petch model is based on dislocation pile-ups at layer interfaces. Stresses at the tip of the pile-up are amplified by the number of dislocations in the pile-up and possesses a mechanical advantage that enables large-scale deformation at low applied stress levels. For smaller thicknesses, $a < 0.5$, where a is the λ - exponent in the empirical Hall–Petch relation, $\sigma = k \lambda^{-a} + k_0$. As the layer thickness is reduced, the number of dislocations in the pile-up is reduced. In the limit where the transfer of slip across interfaces is left to single dislocation, the mechanical advantage is lost requiring large applied stress to accomplish transfer across the interface. This accounts for transition from the Hall–Petch to the plateaus for $\lambda < 50$ nm, and $a \sim 0$, in fig.1. The hardness data of Al–Nb and Cu–Va that is reported in reference⁴⁶ also show similar trends. The differences in the mechanical properties, in particular the Hall–Petch slope and the peak hardness for several bi-material systems as reported in reference⁴⁶ were interpreted in terms of the differences in shear moduli, heat of mixing, and characteristics of interfaces.

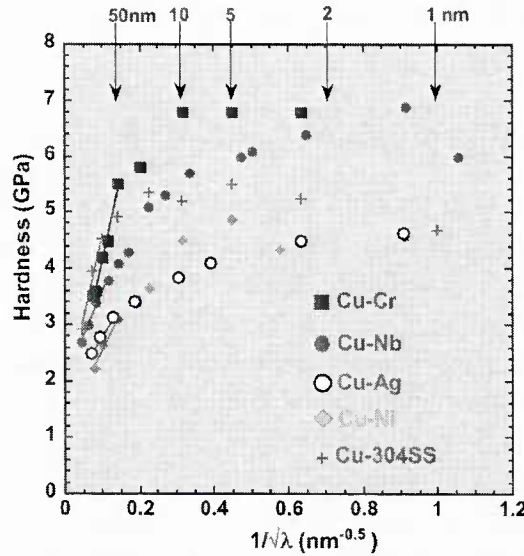


Fig. 1. Hardness as a function of layer thickness for several nanolayered composites (at larger layer thickness the hardness is approximately linear with $\lambda^{-0.5}$) [20].

For λ approximately less than 50 nm, deformation occurs by confined layer slip (CLS) that involves propagation of single dislocation loops parallel to the interfaces in both layers. Dislocations are confined to individual layers since the interface barrier stress to slip transmission is higher than the CLS stress. The CLS stress, which increases as λ decreases, eventually exceeds the interface barrier strength denoted as T^* to the transmission of a single glide dislocation at $\lambda < 5$ nm and the deformation mechanism changes from CLS to interface crossing of single dislocations. T^* is defined as the interface crossing stress, without the mechanical advantage of a dislocation pile-up.

The interface-controlled plasticity (interaction of single dislocations with interfaces) of nanolayered materials constitute a complex problem, because it involves the details of the creation of dislocations at interfaces, their transmission through the interface and processes of storage and relaxation at the interface. An extensive use of theory and modeling (discussed in part in section 2) has been made to elucidate the intricate nature of these complexities.

Hoagland et al developed dislocation models using theory and simulations to interpret the length-scale dependence of strengthening mechanisms in a fcc/bcc Cu-Nb system with incoherent interfaces over a layer thickness ranging from micrometers to less than a nanometer²¹. A dislocation pile-up-based Hall-Petch model was found applicable at the sub-micrometer length scales and the Hall-Petch slope was used to estimate the peak strength of the multilayers. The experimentally measured Hall-Petch slope correlated well with the peak strength of the

multilayer, ~2.6 GPa that was observed at a layer thickness of 1 nm, where the Hall–Petch extrapolation is not valid.

ATOMISTIC MODELING OF NANOSCALE METALLIC INTERFACE STRUCTURES

Atomistic modeling has been used to study interface structure/properties and its effect on the mechanisms of interactions of artificially introduced single dislocations with the interfaces in NMMCs. It has also been used to elucidate details of plastic deformation and the underlying deformation mechanisms during nanoindentation of NMMCs, where dislocations repeatedly nucleate under indenter and then move and interact with interfaces. Medyanik and Shao (2009)⁴⁷⁴⁸ have modeled indentation of a Cu–Ni bilayer with coherent (1 1 1) interface by indenting it both from the Cu and the Ni side. The mechanisms of dislocation– interface interaction observed in the two cases are found to be quite different. Interfacial stacking fault formation is observed only when dislocations propagate from Ni into Cu. In addition to deformation, they also analyze the effects of dislocation–interface interaction on the overall strengthening of the material.

Using molecular dynamics techniques, Saraev and Miller⁴⁹ studied the nano indentation of copper single crystals coated by a thin epitaxial nickel layer forming a semi-coherent interface. They observed that the evolution of plastic deformation depends strongly on the structure of the interface, in particular, on the initial position of misfit dislocations with respect to the indenter. Depending on the position of the misfit dislocations, either dislocation pile-up or dislocation transmission through interface is observed. They also observed a significant strengthening of copper films by thin nickel coatings.

Shao et al⁵⁰ studied dislocation nucleation and propagation during nanoindentation in a Cu–Nb bi-layer with incoherent interface using atomistic simulations. The interface acts as a very strong barrier to dislocation propagation. When dislocations reach the interface from the Cu side, an interfacial shear is observed and no dislocations are transmitted across the interface from Cu into Nb even at very deep indentation depth. However, when indenting from the Nb side, although a considerable amount of interfacial shear occurs, transmission of dislocations was found to occur from Nb to Cu.

As discussed above, atomistic modeling techniques such as molecular dynamics are effective in understanding the effect of interface/structures on the deformation behavior including crack propagation in nanoscale multilayer metallic composites. The next section presents and discusses the mode-I dynamic crack propagation in a Ni–Al bimetallic nanolayer.

DYNAMIC CRACK PROPAGATION IN NANOSCALE NI-AL BILAYER COMPOSITE

Most studies of crack propagation in nano-scale regime have concentrated mainly on the fracture behaviors under tensile loading of either single crystal materials^{51, 52} or of nano-structures containing grain boundaries and interfaces between similar types of materials^{53, 54}. Only few studies have considered crack propagation in nano-structures with interfaces between materials of dissimilar types^{55, 56}. Furthermore, very few studies have been performed at atomistic level to investigate material behaviors under cyclic loading^{57, 58}.

Nanoscale bilayer metallic composites with mismatch in physical and mechanical properties between two metals across the interfaces are frequently encountered in a broad range of applications of technological importance; for example in micro-electro-mechanical and nano-electro-mechanical systems. The mechanical reliability of MEMS/NEMS, in service, depends strongly on their resistance to fracture in the presence of small numbers of cracks formed during their production and operation cycles.

Among various atomistic simulation methods, molecular dynamics (MD) has become a method of choice to study fracture at the atomic scale, as it can provide time-dependent information and allows for the inclusion of strain rate and temperature as meaningful variables in the analysis. In the present work, molecular dynamics simulations is used to investigate crack propagation under cyclic loading in a Ni single crystal and a Ni-Al bi-metallic interface system, in which a crack initiates and propagates from the Ni surface layer towards the Ni-Al bi-metallic interface. This Mode-I dynamic crack growth and propagation under tensile and cyclic loading conditions are discussed.

Modeling Methodology

Molecular dynamics simulations using embedded atom method (EAM) inter-atomic potential were employed to investigate crack propagation in both the Ni and Ni-Al bi-metallic interface system. The selection of the embedded atom method (EAM) for the energy functional in molecular dynamics simulations is a popular choice for the fcc close-packed metals. The EAM potential developed by Pun, et al⁵⁹ was used to define inter-atomic interactions between the Ni-Ni and Ni-Al atoms. For dynamic crack propagation in Ni-Al bilayer composite, it is critical that the potential reproduces the elastic constants as well as surface energies very accurately and the potential used has been fitted to the elastic constants, surface energies and to other bulk and surface properties of Ni and Al.

Bi-Metallic Nanolayer

The schematics and atomistic structures of the simulation geometry used in the present work for the Ni and Ni-Al are shown in figure 2. The x, y and z axes are along the [100], [010] and [001] crystallographic directions, respectively. The (001) [100] crack system was studied in both the configurations. For the (001) [100] crack system, the crack-free surfaces are (001) and the crack propagates along the [100] direction. An initial crack of roughly 1/5 the system length is introduced by partially turning off inter-atomic interactions between atoms in the eight consecutive (001) planes. The two middle planes constitute the upper and lower surfaces of the initial crack. The crack plane is parallel to the *xy* plane. Free boundary conditions were applied in the x and z directions and periodic boundary condition was applied in the y direction (with plane strain condition). Molecular dynamics simulations presented in the work were conducted using molecular dynamics program, LAMMPS⁶⁰.

For the single crystal Ni, the simulation slab had dimensions of $199a_{Ni} \times 7a_{Ni} \times 62a_{Ni}$ with 349,125 atoms, where a_{Ni} (3.52 \AA) is the lattice parameter of Ni. This molecular system configuration is believed to be large enough to take care of the long-range character of the crack strain fields. The Ni-Al bi-layer model was created and assembled from the two semi-infinite perfect crystals of Ni and Al with an orientation relationship of $[100] \parallel [100]$, $[010] \parallel [010]$ and $[001] \parallel [001]$. The two dimensions in the y and z directions were not chosen arbitrarily (due to lattice size mismatch of Ni and Al) but determined such that the strains imposed on the Ni and Al semi-infinite perfect crystals is minimum, and also periodic boundary condition is ensured in the y direction. The total calculated dimensions of $70.015 \times 2.464 \times 21.87 \text{ nm}$ in the three directions in the Ni-Al were found to be comparable with the corresponding three dimensions of $70.048 \times 2.464 \times 21.824 \text{ nm}$ in the Ni single crystal. The energy of the bi-layer was first minimized using conjugate-gradient energy minimization technique. The stresses were then relaxed using MD in NPT ensemble to a pressure of 0 bar and a temperature of 0°K .

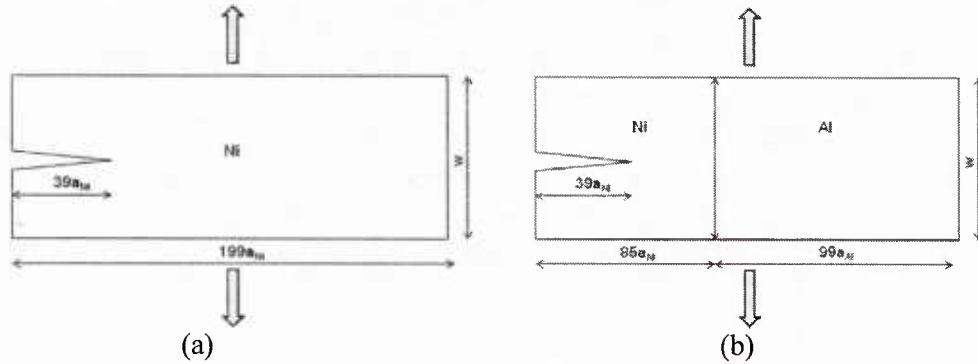


Fig. 2. (a) Schematic of geometry for Ni; (b) Schematic of geometry for Ni-Al bi-layer nanoscale metallic composite.

RESULTS AND DISCUSSIONS

Mode I Uniform Loading

The crack growth and propagation was studied on a (001) plane for both the Ni and Ni-Al. The strain energy release rate (G) is an important quantity in the analysis of crack propagation. This is the amount of energy per unit area that is supplied by the elastic energy stored in the system. It can be calculated by integrating the stress-strain data with respect to strain, ε . In the present molecular strip system, this is given by

$$G = w \int_0^{\varepsilon} \sigma_z(\varepsilon') d\varepsilon' \quad (1)$$

where, w is the width of the strip in the z direction and σ_z is the z component of the stress. The stress for each atom is due to its interaction with all other atoms in the system (within the force cut-off). Atomistic per atom stresses, a stress \times volume formulation, as implemented in LAMMPS were calculated and summed over all the atoms of the system to get σ_z component of the stress. σ_z increases with strain to a certain value and then decreases for all the three systems. The maximum reached value of σ_z was found to be 7.56 GPa for Ni, 4.72 GPa for Ni-Al and 3.69 GPa for Al.

According to Griffith's criteria, a brittle crack under mode I loading propagate when G corresponding to an applied load is equal or greater than $2\gamma_s$, where γ_s is the surface energy of each plane of the crack. The calculated critical strain energy release rate from the stress-strain curve at which the crack starts to propagate in Ni (G_{crit}) is 3.86 J/m² and in Ni-Al (G_{crit}) is 2.4 J/m². The corresponding given values of the Griffith load from the EAM potential, which is twice the (001) surface energy (γ_s), are 3.756 J/m² for Ni (G_{Ni}) and 1.886 J/m² for Al (G_{Al})⁶¹. The snapshot pictures showing an enlarged and a close-up view of the defect structures formed at the crack tip after initiation of plastic deformation at 50 and 70 ps in Ni and Ni-Al are shown in figure 3. The atoms are colored in these figures with yellow for dislocations, brown for stacking faults, and green for surface atoms. The snapshots at 70 ps show formation and evolution of stacking faults associated with nucleation of dislocations from the crack tip. The stacking faults are bounded by dislocation loops, which start at the crack tip. The appearance of dislocations at the crack tip suggests a dynamic brittle-to-ductile transition which leads to a crack arrest in the Ni. When the surfaces of the crack began to roughen atomically, the crack attains a velocity of approximately one third of the Rayleigh wave speed.

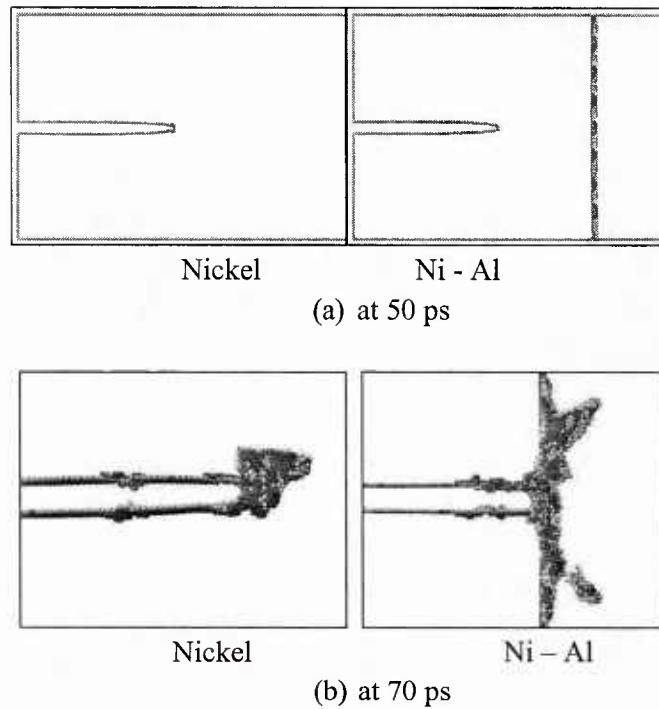


Fig. 3. Snapshots of crack propagation in Ni and Ni-Al Bi-metallic nanolayer

In Ni-Al bimetal system, the crack surfaces initially grow brittle with crack surfaces getting roughened at around one-third of the Rayleigh wave speed. As the crack growth approaches the bi-metal interface, dislocations start emanating from the interfacial bi-layer and they start traveling away from the interface towards the bulk Al. As the crack nears the bi-metal interface, the 'process zone' at the crack tip start interacting with defects at the interface that eventually blunts the crack tip and ceases further crack growth ultimately prohibiting crack from propagating beyond the Ni-Al interface. However, the system continues to dissipate elastic energy through continued creation and motion of dislocations in Al. The snapshots in figure 2(b) for Ni-Al also show formation and evolution of stacking faults associated with nucleation of dislocations from the interfacial bi-layer. The stacking faults, which in this case start at the interfacial layer, are bounded by the dislocation loops (colored in yellow). Further discussions and details are presented in ⁶².

Cyclic Loading

Cyclic loading was applied in a strain-controlled manner at a strain rate of $2.29 \times 10^9 \text{ s}^{-1}$. To simulate fatigue failure in a small number of cycles, the structures were subjected to maximum strains (ϵ_{max}) larger than those required for initiating crack propagation in Ni and Ni-Al. The loading pattern applied to the two systems with a load ratio of 0.85, and two different maximum

applied strains (ϵ_{\max}) of 0.046 is shown in figure 4. A high value of load ratio ($\epsilon_{\min} / \epsilon_{\max} = 0.85$) was used to prevent the inner faces of the crack from contacting each other during unloading. Before applying cyclic load the two systems were subjected to initial tensile strains of 0.039 for ϵ_{\max} of 0.046.

The slabs were initialized at zero temperature and the outward strain rate of $2.29 \times 10^9 \text{ s}^{-1}$ was imposed on the outer most columns of atoms defining the upper free surfaces of the slab in the z direction. A linear velocity gradient was applied across the slab resulting in an increased outward strain with time in the z direction. After loading to a given maximum strain (ϵ_{\max}) the directions of the velocities and the velocity gradient were reversed unloading the system to reach the minimum strain (ϵ_{\min}). The atom velocities were initiated in the required direction at the beginning of each loading and unloading half cycle to alleviate the stress wave overlap that could arise from the high rate of deformation. The loading and unloading cycles lead to the crack growth and propagation and eventual structural failure of the materials.

The crack growth and propagation were studied on the (001) plane for the two systems. Illustrative pictures after various loading cycles (and ϵ_{\max}) showing mechanisms of crack propagation for both Ni and Ni-Al are shown in figure 4. In all of the figures discussed, the atoms are colored according to the centro-symmetry parameter, which is a scalar quantity designed to identify defects such as interfaces, stacking faults and dislocations. In all of the images, atoms with a centrosymmetry parameter close to zero are removed to facilitate easier viewing of the defects inside the structures. The visible atoms are associated with crack surfaces, exterior slab surfaces (only three surfaces are shown), Ni-Al bi-interfacial layer and other defects created during crack propagation. The atoms are colored with yellow for dislocations, brown for stacking faults, and green for surface atoms. The yellow and brown are also associated with atoms with crystallinity other than the fcc.

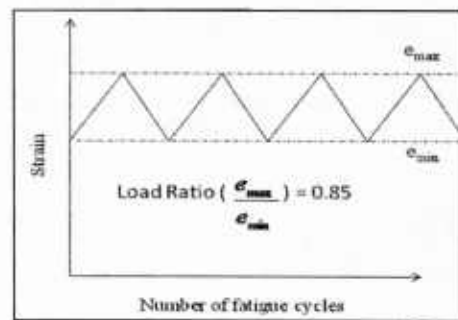


Fig. 4. Strain controlled loading pattern applied to Ni and Ni-Al nanolayer

Case 1: Maximum Strain (ϵ_{\max}) = 0.046

For the maximum applied strain ϵ_{\max} of 0.046, the snapshot sequence of the crack propagation during fatigue cycles 1 and 3 for Ni and Ni-Al (figure 5(a) and 5(b)) show that the crack in both systems at lower ϵ_{\max} move in a straight line with fatigue cleavage of atomic bonds in the crack plane. The crack growth in Ni however, stops after 9 cycles and crack length fluctuates at around 645 angstroms for the next 20 fatigue cycles. The dislocations nucleate from the crack tip during the 29th fatigue cycle. For Ni-Al, the propagating crack reach the interface during the 3rd fatigue cycle. When crack reach the interface, dislocations start emanating from the interfacial bi-layer (figure 5(b)). With continued cyclic loading little changes in the defect structures that form, when crack hit the interface, were observed.

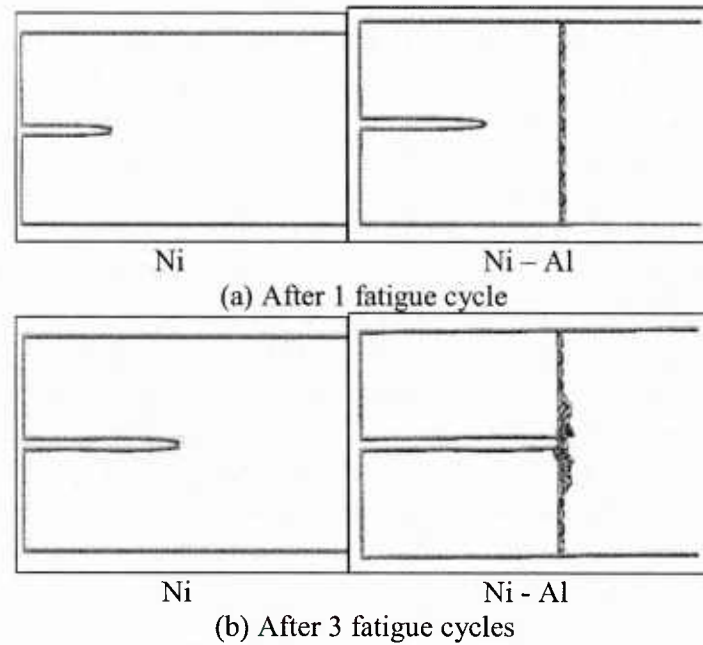


Fig. 5. Crack propagation in Ni and Ni-Al metallic nanolayer (case 1)

Case 2: Maximum Strain (ϵ_{\max}) = 0.057

The snapshot sequence of the crack propagation for the maximum applied strain ϵ_{\max} of 0.057 during loading cycles of 3,7,9 and 10 for Ni and 2,3,4 and 5 for Ni-Al are shown in figures 6 and 7. With higher applied strain, the crack in both Ni and Ni-Al propagate by nucleation of voids in the region near the crack tip. The enhanced plastic deformation at the higher applied strain leads to nucleation of voids in the two systems. In Ni, the dislocations nucleate from the crack tip during the 10th loading cycle as shown in Figure 6 (d) that travels away from the crack tip with continued cyclic loading. In Ni-Al, when the crack reaches the interface during 5th cycle,

dislocations start emanating from the interfacial bi-layer and start traveling away from the interface towards the bulk Al. Figure 7 clearly illustrates this behavior.

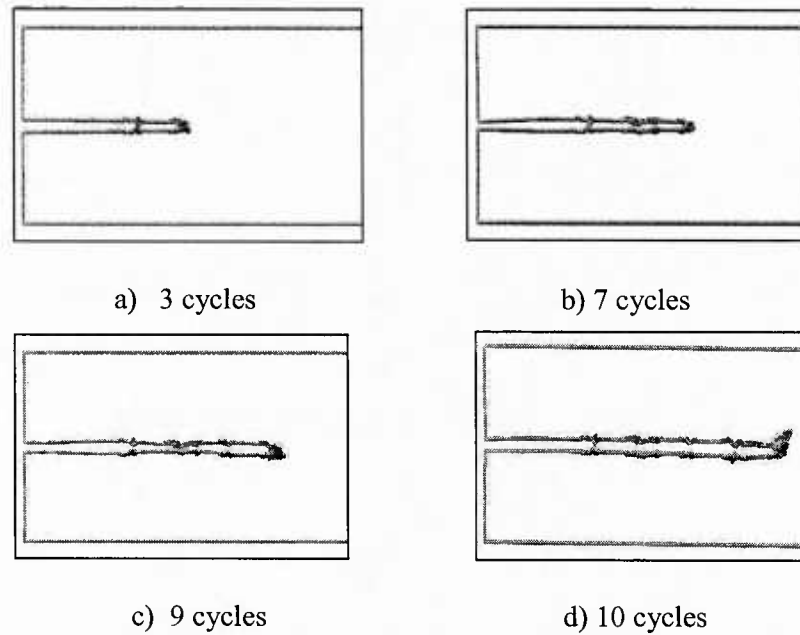


Fig. 6. Crack propagation in Ni single layer (case 2)

The total crack length versus the number of cycles at two different values of the applied maximum strains for both the Ni and Ni-Al are shown in figure 8. The crack in Ni propagates faster when compared to the crack in Ni-Al. However, the crack in both the Ni and Ni-Al at higher applied maximum strain ($e_{\max} = 0.057$) propagates slower when compared to its propagation at lower maximum strain value of e_{\max} (0.046). The present study indicates that the creation of voids at higher maximum strain loading slows down crack propagation in both the Ni and Ni-Al.

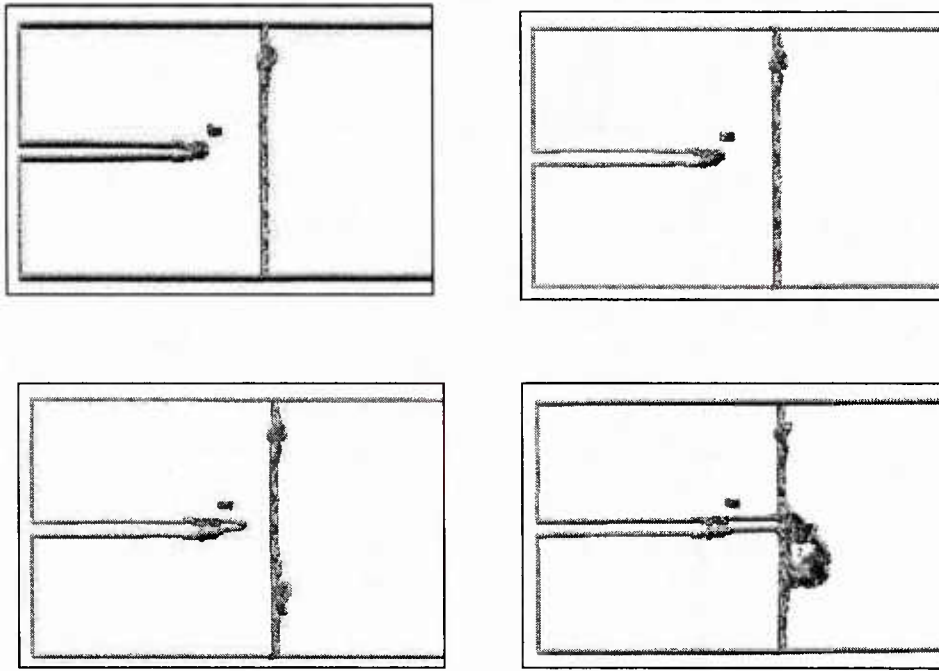


Fig. 7. Crack propagation in Ni-Al bilayer composite

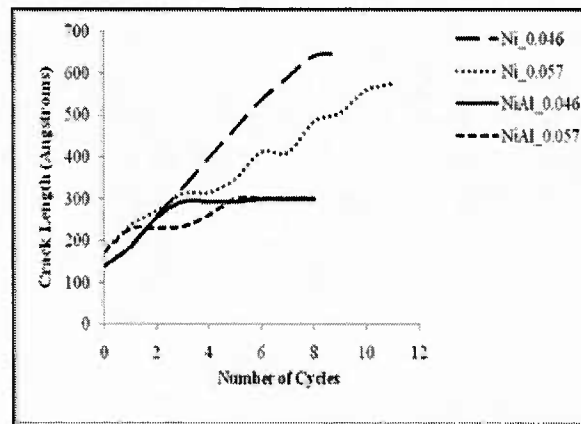


Fig. 8. Crack growth comparison at two different values of maximum applied strain for Ni and Ni-Al bilayer composite

A plot of crack length as it propagates dynamically under cyclic and tensile loading for both the Ni and Ni-Al is shown in figure 9. During tensile loading, plastic deformation around crack tip

dominates crack propagation, resulting in slower crack growth when compared to the crack growth under strain controlled (ϵ_{\max} of 0.046) cyclic loading. The earlier nucleation of dislocations from the crack tip during tensile deformation (at 29 ps), when compared to their nucleation (at 189 ps) during cyclic loading slows down tensile Mode-I crack growth in Ni. In Ni-Al, dislocations nucleate from the crack tip at around 26 ps, which retards crack growth and prevent it from reaching the interface. Further discussions and details are presented in reference

63

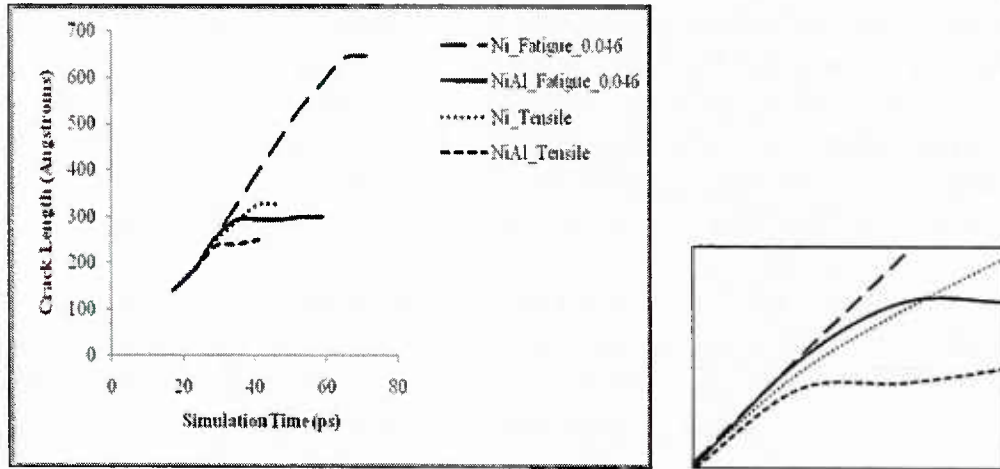


Fig. 9. Dynamic crack growth comparison under uniform tensile and cyclic loading (case 1) for Ni and Ni-Al bilayer composite

CONCLUDING REMARKS

Nanoscale multilayer metallic composites (NMMCs) are of scientific and industrial interest due to their unusual mechanical properties. In considering the structure of NMMCs and its impact on strength and deformation, the role of interfaces becomes clearly paramount. Nanoscale multilayer metallic composites contain extremely high densities of interfaces, and achieve very high strength levels. Interfaces play a crucial role in determining material strength by acting as a strong barrier to slip transmission. The influence of such interfaces on Mode-I crack propagation in a nanoscale bilayer Ni-Al composite employing molecular dynamics (MD) modeling and embedded atom method inter-atomic potential has been investigated and presented.

Results for Ni single crystal are in agreement with predictions given by Abraham, et al.⁵¹ for fcc solids with crack initially growing brittle and eventually undergoing a dynamic brittle-to-ductile transition with a spontaneous proliferation of dislocations from the crack tip following a roughening of the crack surfaces. Results for Ni-Al also showed an initial brittle crack propagation with planar cleavage of atoms between the two neighboring (001) planes defined by

the initial seed crack and crack surfaces getting roughened when the crack propagation speed is about one-third of the Rayleigh wave speed. As the propagating crack approaches the interface, a small bud called the 'process zone' at the crack tip start interacting with interfacial defects that eventually blunts the crack tip and ceases further crack growth.

For the case of cyclic loading, depending on the value of the applied maximum strain, crack propagates either by fatigue cleavage of the atoms in the crack plane or by void nucleation in the regions near the crack tip. In Ni-Al, as crack approaches the bi-metallic interface, dislocations start emanating from the interfacial bi-layer. The presence of interface in the Ni-Al prohibit crack from propagating beyond the interface. The creation of voids slows down crack growth in both the Ni and Ni-Al at higher value of ϵ_{\max} during cyclic loading. Plastic deformation dominates crack propagation during tensile loading that result in slower crack growth, when compared to the crack growth under cyclic loading. The earlier nucleation of dislocations at the crack tip in Ni-Al prevents crack from reaching the interface during tensile loading.

In summary, presence of semi-coherent interface in the nanoscale Ni-Al bilayer composite was found to prohibit crack from propagating beyond the interface. An understanding of interface effects on fracture on NMMCs is essential in forming a critical foundation for the development of newer generations of nanoscale multilayer metallic composite structural materials with better combination of properties.

REFERENCES

1. A. Misra, M. Verdier, Y. C. Lu, H. Kung, T. E. Mitchell, M. Nastasi and J. D. Embury, *Scripta Materialia* 39 (4/5), 555 (1998)
2. B. M. Clemens, H. Kung and S. A. Barnett, *MRS Bull* 24, 20 (1999)
3. A. Misra and H. Kung, *Adv. Eng. Mater.* 3, 217 (2001)
4. M. A. Phillips, B. M. Clemens and W. D. Nix, *Acta Materialia* 51, 3157 (2003)
5. A. Misra, H. Kung and J. D. Embury, *Scripta Materialia* 50, 707 (2004)
6. A. Misra, J. P. Hirth, R. G. Hoagland, J. D. Embury and H. Kung, *Acta Materialia* 52, 2387 (2004)
7. A. Misra, X. Zhang, D. Hammon and R. G. Hoagland, 53, 221 (2005)
8. P. M. Anderson, J. F. Bingert, A. Misra and J. P. Hirth, *Acta Materialia* 51, 6059 (2003)
9. A. Misra, R. G. Hoagland and H. Kung, *Philosophical Magazine* 84 (10), 1021 (2004)
10. M. N. Baibich, J. M. Broto, F. Fert, V. D. Nguyen and F. Petroff, *Phys. Rev. Lett.* (61), 2472 (1988)
11. C. Montcalm, B. T. Sullivan, M. Ranger, J. M. Slaughter, P. A. Kearney and C. M. Falco, *OPTICS LETTERS* 19 (13), 1004 (1994)
12. W. Schwarzacher and D. S. Lashmore, *IEEE TRANSACTIONS ON MAGNETICS* 32 (4), 3133 (1996)

13. S. PalDey and S. C. Deevi, *Materials Science and Engineering A* 342, 58 (2003)
14. A. R. Maligno, D. Whalley and V. V. Silberschmidt, *Materials Science and Engineering* 10, 012087 (2010)
15. E. Pellicer, A. Varea, S. Pane', B. J. Nelson, E. Mene'ndez, M. Estrader, S. Surin'ach, M. D. Baro', J. Nogue's and J. Sort, *Adv. Funct. Mater.* 20, 983 (2010)
16. S. I. Rao and P. M. Hazzledine, *Philosophical Magazine A* 80 (9), 2011 (2000)
17. R. G. Hoagland, T. E. Mitchell, J. P. Hirtha and H. Kunga, *Philosophical Magazine A* 82 (4), 643 (2002)
18. A. Misra, J. P. Hirth and H. Kung, *PHILOSOPHICAL MAGAZINE A* 82 (16), 2935 (2002)
19. C. H. Henager Jr., R. J. Kurtz and R. G. Hoagland, *Philosophical Magazine* 84 (22), 2277 (2004)
20. R. G. Hoagland, R. J. Kurtz and C. H. Henager Jr., *Scripta Materialia* 50, 775 (2004)
21. A. Misra, J. P. Hirth and R. G. Hoagland, *Acta Materialia* 53 4817 (2005)
22. R. G. Hoagland, J. P. Hirth and A. Misra, *Philosophical Magazine* 86 (23), 3537 (2006)
23. F. Akasheh, H. M. Zbib, J. P. Hirth, R. G. Hoagland and A. Misra, *JOURNAL OF APPLIED PHYSICS* 102, 034314 (2007)
24. M. J. Demkowicz, J. Wang and R. G. Hoagland, 14, 141 (chapter 83) (2008)
25. K. Al-Fadhalah, *Phil. Mag.* 85, 1419 (2005)
26. M. J. Demkowicz and R. G. Hoagland, *Journal of Nuclear Materials* 372, 45 (2008)
27. A. Misra, M. J. Demkowicz, J. Wang and R. G. Hoagland, *JOM Journal of the Minerals, Metals and Materials Society* 60 (4), 39 (2008)
28. J. Wang, R. G. Hoagland, J. P. Hirth and A. Misra, *Acta Materialia* 56, 5685 (2008)
29. J. Wang, R. G. Hoagland, J. P. Hirth and A. Misra, *Acta Mater.* 56, 3109 (2008)
30. J. Wang, R. G. Hoagland and A. Misra, *J Mater. Res.* 23, 1009 (2008)
31. J. Wang, R. G. Hoagland and A. Misra, *Scripta Materialia* 60, 1067 (2009)
32. J. Wang and A. Misra, *Current Opinion in Solid State and Materials Science* 15, 20 (2011)
33. I. N. Mastorakos, H. M. Zbib and D. F. Bahr, *APPLIED PHYSICS LETTERS* 94, 173114 (2009)
34. J. P. Hirth and X. Feng, *J. Appl. Phys.* 67, 3343 (1990)
35. J. McKeown, A. Misra, H. Kung, R. G. Hoagland and M. Nastasi, *Scripta Mater* 46, 593 (2002)
36. G. V. Kurdjumov and G. Sachs, *Z. Phys.* 64, 325 (1939)
37. R. Asaro, *Advances in Applied Mechanics* 23, (1983)
38. C. L. Kelchner and S. J. Plimpton, *Physical Review B* 58, 11085 (1998)
39. P. M. Anderson and C. Li, *Nanostruct Mater.* 5, 349 (1995)
40. L. H. Friedman and D. C. Chrzan, *Phys. Rev. Lett.* 81, 2715 (1998)
41. H. B. Huang and F. Spaepen, *Acta Materialia* 48, 3261 (2000)

42. P. M. Anderson and Z. Li, *Mater Sci Eng A* 319, 182 (2001)
43. Y. Shen and P. M. Anderson, *Acta Mater* 54, 3941 (2006)
44. J. D. Embury and J. P. Hirth, *Acta Metall Mater.* 42, 2051 (1994)
45. P. M. Anderson, T. Foecke and P. M. Hazzledine, *MRS Bull.* 24, 27 (1999)
46. E. G. Fu, N. Li, A. Misra, R. G. Hoagland, H. Wang and X. Zhang, *Materials Science and Engineering A* 493, 283 (2008)
47. S. N. Medyanik and S. Shao, *Comput. Mater. Sci.* 45, 1129 (2009)
48. S. Shao and S. N. Medyanik, *Mechanics Research Communications* 37, 315 (2010)
49. D. Saraev and R. E. Miller, *Acta Materialia* 54, 33 (2006)
50. S. Shao and S. N. Medyanik, *Modelling Simul. Mater. Sci. Eng.* 18, 055010 (2010)
51. F. F. Abraham, D. Brodbeck, R. A. Rafey and W. E. Rudge, *Phy. Rev. Lett.* 73, 272 (1994)
52. T. Zhu, J. Li and S. Yip, *Physical Review Letters* 93, 025503 (2004)
53. H. V. S. D. Farkas, P.M. Derlet, *Physical Review B* 66, (2002)
54. J. F. B. R.E. Rudd, *Comp. Mater. Sci.* 24, (2002)
55. M. J. Buehler and H. Gao, *Strength, Fracture and Complexity* 3, 105 (2005)
56. K. Gall, M. F. Horstmeyer, M. V. Schilfgaarde and M. I. Baskes, *Journal of Mechanics and Physics of Solids* 48, 2183 (2000)
57. D. Farkas, M. Willemann and B. Hyde, *Physical Review Letters* 94, 165502 (2005)
58. G. P. Potimiche, M. F. Horstmeyer, P. M. Gullett and B. Jelinek, *Proceedings of the Royal Society A* 462, 3707 (2006)
59. G. P. Purja Pun and Y. Mishin, *Philosophical Magazine* 89, 3245 (2009)
60. S. J. Plimpton, *Journal of Computational Physics* 117, 1 (1995)
61. Y. Mishin, D. Farkas, M. J. Mehl and D. A. Papaconstantopoluos, *Physical Review B* 59, 3393 (1999)
62. Y. Purohit and R. Mohan, *Molecular Dynamics of Crack Propagation in Nickel and Nickel-Aluminum Bilmetal Interface*. In *International Mechanical Engineering Congress and Exposition (IMECE)*, Americal Society of Mechanical Engineers, VanCouver, Canada, (2010), Vol. IMECE2010-38677,
63. Y. Purohit and R. Mohan, *Molecular Dynamics of Crack Growth in Nickel and Nickel-Aluminum Bi-Metallic Interface System Under Cyclic Loading*. In *2011 ASME International Mechanical Engineering Congress and Exposition*, (ASME), A. S. f. M. E., Ed., Americal Society for Mechanical Engineers, Denver, Colorado, USA, (2011), Vol. IMECE 2011-65150,
64. Y. Liang, Q. Han and J. Ou, *Journal of Computational and Theoretical Nanoscience* 11, 71 (2014)
65. M. M. S. Fakhrabadi, B. Dadashzadeh, V. Norouzifard and A. Allavedizadeh, *Journal of Computational and Theoretical Nanoscience* 10, 1921 (2013)
66. M. H. Msazadeh and K. Dehghani, *Journal of Computational and Theoretical Nanoscience* 10, 1497 (2013)

67. M. Xiang, J. Cui, X. Tian and J. Chen, Journal of Computational and Theoretical Nanoscience 10, 1215 (2013)
68. M. E. Kilic and S. Erkoc, Journal of Computational and Theoretical Nanoscience 10, 104 (2013)
69. M. E. Kilic and S. Erkoc, Journal of Computational and Theoretical Nanoscience 10, 112 (2013)
70. N. Nouri and S. Ziaei-Rad, Journal of Computational and Theoretical Nanoscience 9, 2144 (2012)
71. R. Mohan, Y. Purohit and Y. Liang, Journal of Computational and Theoretical Nanoscience 9, 649 (2012)

B: COMPUTATIONAL ENABLING TECHNOLOGIES

Section B focuses on the research activities related to the enabling technologies. In particular, multi-scale modeling approaches are required to accurately capture the disparate length scale effects in various engineering problems. Project work in this area focused on the coupled Lattice Boltzmann and Navier Stokes modeling for flow problems in collaboration with University of Alabama at Birmingham. Further developments in these concurrent coupled modeling developments are needed. The present efforts are geared towards applications in understanding the nano fiber, nano tube resin flow interactions in composites material processing. Due the low length scale size of nanofibers in comparison to the resin flow domain, low length scale methods in the vicinity of the nanofiber flow region and correlation with the macroscopic flow field. Research and modeling investigations and modeling investigations comparing the Lattice Boltzmann and Navier Stokes approaches are presented in the sub-section B-1.

High performance computing architectures are evolving over the years with the Graphical Processing Units (GPU) are providing superior performances for computationally intensive problems. Recent research efforts involved the porting and implementation of the computational process flow process modeling developments on a GPU cluster are presented in sub-section B-2.

Physics based flow modeling provides an effective way to simulate the resin infusion process in liquid composite molding processes for polymer composite structures. These are effective to provide optimal injection time and locations for given process parameters of resin viscosity and preform permeability prior to resin gelation. However, there could be significant variations in these two parameters during actual manufacturing due to differences in the resin batches, mixes, temperature, ambient conditions for viscosity; in the preform rolls, compaction, etc., for permeability. Research to understand the influence of uncertainties in these parameters on the resin infusion time was initiated via a probabilistic, non-deterministic modeling methodology using deterministic resin flow modeling and statistical analysis are presented in section B-3.

B-1 Multi-Scale Simulation Investigations of Nanofiber Resin Interactions using Lattice Boltzmann Equations and Finite Volume Methods

In collaboration with University of Alabama at Birmingham (UAB)

Authors: Y. H. Kim (UAB), R. Mohan (NCAT), R. Koomulli (UAB), B. Soni (UAB)

The orientation/distribution of carbon nanotube (CNT) and other nanofibers in polymer matrix, one of main factors in manufacturing high-performance multifunctional composites, is an

important aspect to be considered during the development of new CNT composites with enhanced mechanical, electrical and thermal properties. However, the disparate length scales involved and mechanical properties of nanotube and rheological properties of polymer matrix around CNT and nanofibers hinder researchers from elucidating the problem via computational modeling. Understanding this problem requires a multi-scale computational approach. Different computational solvers for each of these scales, bridging techniques between the solvers, and a representative model of a carbon nanotube/nanofiber are needed for the simulation of this class of multi-scale and multi-disciplinary problems.

Project efforts towards this objective focused on 1) the coupling of a macro-scale solver, HYB3D, and a meso-scale solver, Regularized Lattice Boltzmann (LB) equation solver, for computational fluid dynamics, 2) the generation and analysis of a representative volume element for CNT using elastic theories and ANSYS as computational structural dynamics code, and 3) the handling of moving boundaries in lattice cell for fluid structure interaction using simple standard bounce-back boundary schemes. A 3D flow past a circular cylinder is simulated using Bhatnagar-Gross-Krook dynamics and regularized LB methods as a demonstration of the LB method. The comparison of the results between two models (macro finite volume solver HYB3D and meso solver LB) demonstrates that the regularized LB method can be used for coupling meso-scale and macro-scale solvers.

Recently, carbon nanotubes (CNT) are used as filler in polymer composites because of its dramatic physical properties including mechanical strength [1-3], electrical conductivity and capacity [4], and thermal conductivity [5]. These remarkable properties make CNT as one of the most promising reinforcing materials in the fabrication of advanced polymer composites [6-9]. Although the properties of CNT polymer composites [1-9] and micro scale fiber behaviors in graphite composite manufacturing [10-13] has been reported in the literature, the orientation and configuration of CNTs in a polymer resin flow during the manufacturing process have rarely been studied. This is due to the requirement of a comprehensive analysis of fluid structure interaction (FSI) between CNTs and polymer, interaction between CNTs, and electrical/chemical interactions that need to be considered in the experimental and computational approaches for the analysis of this problem. The limitations of experimental facilities and approaches to study this multi-scale problem lead researchers to investigate computational simulations. Researchers have developed numerical schemes on the multi-scale simulation methods for suspension flow, models for CNTs using elastic theories, and numerical methods for handling of moving boundaries in FSI for solving these types of problem.

Typically, the orders of magnitudes of length and time scales in CNT composite simulations can span from 5 to 12. Even with the recent advanced computer systems and algorithms, it is impractical to analyze the phenomena of CNT composites with computational simulation using a

single scale due to the wide span of these length and time scales. Inevitably, multi-scale methods are required to overcome this problem. These multi-scale computational methods can be categorized into two groups: sequential method and concurrent method. The sequential method involves critical information pass from a lower length scale (for example, molecular scale) to a higher length scale (for example, macroscale). This method is a proper approach when an effective model in molecular scale can entirely be employed for input parameters in the continuum constitutive model. Thus, this method can be applied to analyze polymer composites which usually consist of fluid (polymer matrix) and solid (fillers). The concurrent method involves direct coupling between different scales. This is an appropriate approach when important atomic scale phenomena are focused on localized space, such as at a crack tip, grain boundary, or nano-indenter [14].

The sequential multi-scale method is a proper method for CNT polymer composites. The main barrier in this method is the development of an efficient and accurate way of bridging different scales. The micro scale methods for bridging nano to micro scales include Brownian dynamics (BD), Dissipative particle dynamics (DPD), Lattice Boltzmann (LB), Time-dependent Ginzburg-Landau (TDGL), and Dynamic density functional theory (DDFT). Due to the range of scales in time and length of CNT composites during process flow interactions is limited to micro and macro scales, molecular effects are neglected in the present study.

Brownian Dynamics (BD) [15] employs an implicit continuum solvent description instead of explicit solvent description in molecular dynamics (MD) by assuming no internal motions of molecules. This assumption allows much larger time order than that of MD. Therefore, the BD is proper method incorporating slow suspension flow of mixed polymer and solvent including fast motions of solvent molecules. Due to the approximation of the fast degrees of freedom by fluctuating forces in BD, the energy and momentum is not conserved. This non-conservation causes consequently the composite system not to be hydrodynamic in macroscopic scale. Thus, this BD method cannot be bridged with the Navier-Stokes equations.

Discrete Particle Dynamics (DPD) [16] is a particle-based method like MD. DPD handle the particle at micro scale different from MD at molecular scale. The potentials between particles are approximated in DPD using simple order basis function at microscopic length scale. The conserved force and momentum at micro scale enable this method to incorporate hydrodynamic equation such as Navier-Stokes equation at macro scale. The energy, however, is not conserved in this method due to the presence of dissipative and random forces.

LB [17] method is originated from discretized, simplified and fictitious molecular dynamic lattice gas automation. This method is usually employed to investigate phase separation of binary fluid in the existence of filler particles in polymer composites. An important advantage of LB

method is that the interactions between particles at micro scale can easily be incorporated into numerical model such as Navier-Stokes equations at macro scale. In the LB method, particle occupation variables are replaced by single-particle distribution functions. In addition, individual particle motion and interaction between particles in the kinetic equations are neglected. This assumption causes this method to be numerically unstable and consequently may lead to unreasonable physical results in the case of high force interaction between particles.

TDGL generalized Cahn-Hilliard-Cook nonlinear diffusion equation for a binary blend into phase-field and reaction-diffusion between blended polymer and fillers [18]. By minimizing free-energy function in this method, time-dependent structural evolution of the blended polymer is investigated. A simplified version called cell dynamic method, of the TDGL method has been developed by Oono et al. [19] by replacing Laplacian term with isotropic discretized counterpart. Both methods have been recently and widely used to analyze the phase-separation of nanocomposites [20-22].

DDFT method integrates Gaussian mean-field statics into TDGL method to model the behavior of polymer fluid implemented in MesodynTM from Accelrys [23]. The integration enables this method to employ numerically full polymer path without truncating free energy at a certain level. In addition, this method has a capability of simulating viscoelastic properties of polymer fluid.

Of all these different methods, LB method has been widely used for simulating particles/fibers in suspension flows because of the ease of generation of the mesh, data locality for parallelization, flexible boundary conditions, and noise-free solution. Ladd [24, 25] provided theoretical foundation and applications of a general technique for simulating solid-fluid suspension via discrete Boltzmann equation. Lallemand and Luo [26] developed an LB method for moving boundaries for analyzing moving cylinder in a transient Couette flow. Lee-Edwards boundary conditions for sheared suspension flow in LB method were used by Lorenz and Hoekstra [27] to capture shear-thickening behaviors. Also, particle-particle interactions in shear flow were analyzed using a chain like cluster of suspended particles by Hyvaluoma et al. [28]. Joshi and Sun [29] developed multiphase LB method for particle suspensions. Ramachandran et al. [30] developed an LB model for suspensions of self-propelling colloidal particles via active particle with velocity field. While aforementioned models/schemes are applied to rigid bodies, Wu et al. [31, 32], MacMeccan et al. [33], Buxton et al. [34], Lorenz et al. [35] and Dupin et al. [36] simulated deformable particles using schemes for LB method to handle moving boundaries (fluid-solid interface) and/or models for representing particles/fibers.

The schemes for handling FSI interface due to the deformable or moving rigid particles/fibers have been investigated and reported in the literature. The bounce-back scheme for no-slip velocity boundary conditions at walls has been most widely used due to ease in implementation, although the scheme has only a first order accuracy at the boundaries [37, 38]. This simple boundary scheme is used to analyze fluid flows in complicated geometries such as flow through

porous media, flow around high curvature boundary, etc. This mismatch in the order of accuracy in the LB method degrades the accuracy of entire results [33, 34, 36]. For more accurate results in the complex geometries, the method has been improved using interpolation schemes including spatial linear [39], quadratic [40], and multi-reflection [41] methods. Wu et al. developed an immersed boundary LB scheme [31] and external boundary force [32] on the interface between fluid and structure without the interpolation schemes.

To model the micro or nano structures in polymers, researchers have used several representative models ranging from simplified models such as spring models [34, 36] to equivalent-continuum approach (ECA) or self-similar approach (SSA) [42-44] combining with MD to consider local interaction loading forces between molecules by their potential energies including covalent bond stretching, bond-angle bending, and Van der Waals interactions. The foundation of ECA is to develop a representative volume element (RVE) at macroscopic length scale to statistically represent the local interaction between microscopic elements. The RVE has been developed ensuring that the element length scale is consistent with the smallest constituent that covers the RVE continuum behavior properly. The developed RVE is then used iteratively or periodically at macro scale. The RVE models have usually the following basic assumptions: (a) linear elastic properties, (b) the identical fillers in shape and contents, (c) no slip, crack and de-bonding between polymer matrix and fillers. Based on these assumptions, the RVE is described as multi-material elements using volume fraction. The Halpin-Tsai [45] and Mori-Tanaka [46] models are widely used in polymer composites for this method in micro scale. These RVE models have expanded to nano scale modifying the basic assumptions. Li et al. [47] studied CNT epoxy composite strength using Halpin-Tsai and Mori-Tanaka models. Gao and Li [48] developed a shear-lag model to predict the interfacial stress of CNT composite using RVE. Liu and Chen [49,50] employed FEM and boundary element method (BEM) to study CNT composite using RVEs containing CNTs modeled as thin elastic layer for short CNT or an open cylinder for long CNT. Liu et al. [51] recently developed BEM models combined with a new cohesive interface model with MD and analyzed Young's modulus of CNT composite. Tserpes et al [52] employed a multi-scale RVE to investigate the effect of interfacial shear strength on the tensile behavior of CNT composites. Pantano et al. [53] studied the effect of CNT curvature and interface interaction with polymer matrix on composites using RVE and FEM.

From the literature search, it was concluded that the Lattice Boltzmann (LB) method is suitable for the modeling of micro-scale behavior and continuum modeling is suitable for simulation mean suspension flow. Based on the conclusion, the OpenLB for LB method and HYB3D for continuum modeling were chosen. The sequential multi-scale coupling method of two solvers was chosen and the part of coupling procedures involving mesh generation, code modification and development for space and time synchronization has been implemented. The governing equation of HYB3D was studied to find out the passing parameters for the coupling procedures

with OpenLB and the brief introduction of the code was introduced. The validation of Poiseuille flow through square duct was shown in the last annual report with an LB solver and an in-house Navier-Stoke (NS) solver to couple the meso-scale LB and macro-scale NS solvers. The results showed good agreement with the analytical solution. In the coupling procedures of two solvers, the distribution function updated by moving boundaries in OpenLB is related to passing parameters. In the present project work, the validation of flow past moving cylinder in a channel flow at rest was performed using a moving boundary algorithm. The results from fixed frame showed lots of fluctuation, which may be caused by taking averaged values of the nearest nodes in the extrapolation of distribution function from solid to fluid region due to the moving boundaries. To remove the fluctuation, the other methods for the extrapolation schemes could be used.

Lattice Boltzmann Method

The LB equation has been originated from lattice gas (LG) automata employing a discrete kinetics with a discrete lattice and time or expansion of the continuum Boltzmann equation with a discrete set of velocities for small Mach number [1]. Frishch et al [2] found LB equation to be turned into NS equation using Lattice Bhatnagar-Gross-Krook (BGK), Chapman-Enskog expansion, which is a formal multiscaling expansion method.

In this sector, the details of derivation of LB equation from lattice gas automata, Lattice BGK from LB, and NS equation from LB equation within small Knudsen and Mach number will be described.

Lattice Boltzmann Equation from Lattice Gas Automata

Let's define a set of Boolean variables $\delta_i(\mathbf{x}, t)$ ($i = 1, \dots, N$), describing the particle population function on nodes of lattices, where N is the number (6 in hexagonal lattice in Figure 1) of direction of the discrete velocities at each node. The evolution equation of LG automata can be written as:

$$\delta_i(\mathbf{x} + \mathbf{e}_i, t + 1) = \delta_i(\mathbf{x}, t) + \Omega_i(\delta_i(\mathbf{x}, t)), \quad (i = 1, \dots, M), \quad (1)$$

where \mathbf{e}_i are the local particle velocities, Ω_i is a collision operator. The movement of particles in the evolution can be separated into streaming (propagation) and collision phases. The major disadvantage of using LG automata method for macroscopic flow applications is the occurrence of the statistical noises at each node, which has almost no effect on dynamics of the flow and could be prone to lead divergence of solution in hydrodynamic problems. For solving these problems, LB method have been developed using the averaged velocity distribution function, which represents each group of particle movements by neglecting the individual particle movements and interaction between themselves. The LB equation can be written with lattice

units (lattice space (Δx) and time (Δt) increments are set to be unitary) by replacing Eq. (1) with $f_i = \langle \delta_i \rangle$, where $\langle \rangle$ denotes an ensemble average operator, and $f_i \equiv f_i(x, t)$ is the averaged velocity distribution function:

$$f_i(x + e_i, t + 1) = f_i(x, t) + \Omega_i(f_i(x, t)), \quad (i = 1, \dots, N), \quad (2)$$

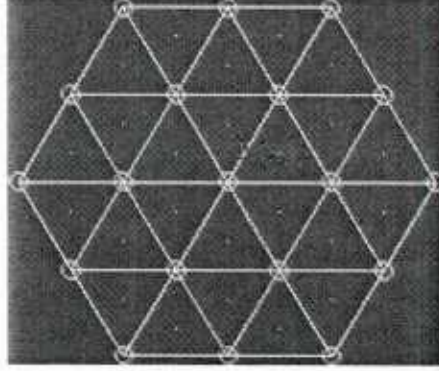


Figure 1. A particle population function on a node of hexagonal lattices

Lattice *Bhatnagar-Gross-Krook* (BGK) from Lattice Boltzmann Equation

If fluid has long wave length and low frequency properties (small Knudsen number) which can be covered by averaged velocity distribution function, the Δx and Δt in Eq. (2) can be considered as small parameter ε . The left hand side of Eq. (2), then, can be expanded using Taylor expansion series in time and space to second order in ε :

$$\frac{\partial f_i}{\partial \tau} + \nabla \cdot e_i f_i + \frac{1}{2} \varepsilon \left(\nabla \nabla : e_i e_i f_i + 2 \nabla \cdot e_i \frac{\partial f_i}{\partial \tau} + \frac{\partial^2 f_i}{\partial \tau^2} \right) = \frac{\Omega_i}{\varepsilon} \quad (3)$$

Similarly, the averaged distribution function f_i can be defined as:

$$f_i = f_i^{\text{eq}} + \varepsilon f_i^{\text{neq}} \quad (4)$$

where, f_i^{eq} is denoted as an equilibrium distribution function, and f_i^{neq} is defined as a nonequilibrium distribution function which can be expanded by low Mach number expansion [1] as:

$$f_i^{\text{neq}} = f_i^{(1)} + \varepsilon f_i^{(2)} + O(\varepsilon^2) \quad (5)$$

where, the superscripts of f_i are the order of the Mach number expansion.

To linearize the collision operator Ω_i , after inserting f_i in Eq. (4) into Ω_i , it can be expanded with Taylor series as:

$$\begin{aligned}\Omega_i(f) = \Omega_i(f^{eq}) + \varepsilon \frac{\partial \Omega_i(f^{eq})}{\partial f_j} f_j^{(1)} \\ + \varepsilon^2 \left(\frac{\partial \Omega_i(f^{eq})}{\partial f_j} f_j^{(2)} + 2 \nabla \cdot \mathbf{e}_i \frac{\partial f_i}{\partial \mathbf{r}} + \frac{\partial^2 \Omega_i(f^{eq})}{\partial f_j \partial f_k} f_j^{(1)} f_k^{(1)} \right) + O(\varepsilon^3)\end{aligned}\quad (6)$$

When ε goes to the zero asymptotically, Eq. 6 can be written as:

$$\frac{\Omega_i(f)}{\varepsilon} = \frac{M_{ij}}{\varepsilon} (f_j - f_j^{eq}) \quad (7)$$

where $M_{ij} \equiv \frac{\partial \Omega_i(f^{eq})}{\partial f_j}$ is called as the collision matrix [3], which represents the scattering rate between two arbitrary directions i and j . In the collision, mass and momentum should be conserved so that M_{ij} should satisfy following equations

$$\sum_{i=1}^M M_{ij} = 0 \quad (8)$$

$$\sum_{i=1}^M M_{ij} \mathbf{e}_i = 0 \quad (9)$$

Assuming M_{ij} is relaxed to an equilibrium state at a single rate τ , M_{ij} is expressed as:

$$M_{ij} = -\frac{1}{\tau} I \quad (10)$$

where, I is identity matrix. The new defined M_{ij} also should satisfy Eq.'s (8) and (9), then consequently local equilibrium of macroscopic parameters involving density ρ and momentum $j = \rho u$, should be conserved as:

$$\rho = \sum_{i=1}^M f_i \quad (11)$$

$$j = \sum_{i=1}^M f_i \mathbf{e}_i \quad (12)$$

In addition, similarly, the $\Omega_i \equiv \Omega_i(f_i(\mathbf{x}, t))$ needs to satisfy conservation of total mass and momentum at the local lattice as follows:

$$\sum_{i=1}^M \Omega_i = 0 \quad (13)$$

$$\sum_{i=1}^M \Omega_i \mathbf{e}_i = 0 \quad (14)$$

Inserting Eq. (10) into Eq. (7), the lattice Bhatnagar-Gross-Krook (BGK) collision term [4] are derived as:

$$\frac{\Omega_i(f)}{\varepsilon} == -\frac{1}{\tau} f_i^{\text{neq}}, \quad (15)$$

and the lattice BGK equation with Eq. (15) is defined as:

$$f_i(\mathbf{x} + \mathbf{e}_i, t + 1) = f_i(\mathbf{x}, t) - \frac{(f_i - f_i^{\text{eq}})}{\tau} \quad (16)$$

Navier-Stokes Equation from Lattice Boltzmann Equation

The formal multiscaling expansion, Chapman-Enskog is required to derive a macroscopic NS equation from LB equation, assuming that the time scale τ_2 is much smaller than time scale τ_1 [5]. The expansion of time and space derivative is written as:

$$\frac{\partial}{\partial t} = \varepsilon \frac{\partial}{\partial \tau_1} + \varepsilon^2 \frac{\partial}{\partial \tau_2} + O(\varepsilon^3) \quad (17)$$

$$\frac{\partial}{\partial \mathbf{x}} = \varepsilon \frac{\partial}{\partial \mathbf{x}_1} + O(\varepsilon^2) \quad (18)$$

Using low Mach number expansion used in Eq. (5), the collision operator Ω_i can similarly be expanded as:

$$\Omega_i = \varepsilon \Omega_i^{(1)} + \varepsilon^2 \Omega_i^{(2)} + O(\varepsilon^3) \quad (19)$$

Applying the above expansions to Eq. (6), then the scale separated version of the equation with neglecting $O(\varepsilon^3)$ is obtained as:

$$\varepsilon \Omega_i^{(1)} + \varepsilon^2 \Omega_i^{(2)} = \left(\varepsilon \frac{\partial}{\partial \tau_1} + \varepsilon^2 \frac{\partial}{\partial \tau_2} + \varepsilon \nabla_1 \cdot \mathbf{e}_i + \frac{1}{2} \varepsilon^2 \frac{\partial^2}{\partial \tau_2^2} + \varepsilon^2 \frac{\partial}{\partial \tau_1} \nabla_1 \cdot \mathbf{e}_i + \frac{1}{2} \varepsilon^2 \nabla_1 \nabla_1 : \mathbf{e}_i \mathbf{e}_i \right) (f_i^{\text{eq}} + \varepsilon f_i^{(1)}) \quad (20)$$

Considering collision operator works only on f_i^{neq} in Eq. (13), Eq. (11) can be written as:

$$\rho = \sum_{i=1}^M f_i = \sum_{i=1}^M f_i^{\text{eq}} \quad (21)$$

Eq.'s (11) and (13) play an important role in leading LB to NS equation. Expanding Eq. (13) on two different orders ε and ε^2 in Eq. (20), the following equations are given as:

$$\sum_{i=1}^M \Omega_i^{(1)} = \frac{\partial}{\partial \tau_1} \sum_{i=1}^M f_i^{\text{eq}} + \nabla_1 \cdot \sum_{i=1}^M \mathbf{e}_i f_i^{\text{eq}} = \frac{\partial}{\partial \tau_1} \rho + \nabla_1 \cdot \mathbf{j}^{(0)} = 0 \quad (22)$$

and

$$\begin{aligned}
\sum_{i=1}^M \Omega_i^{(2)} &= \frac{\partial}{\partial \tau_1} \sum_{i=1}^M f_i^{(1)} + \frac{\partial}{\partial \tau_2} \sum_{i=1}^M f_i^{\text{eq}} + \nabla_1 \cdot \sum_{i=1}^M \mathbf{e}_i f_i^{(1)} + \frac{1}{2} \frac{\partial^2}{\partial \tau_1^2} \sum_{i=1}^M f_i^{\text{eq}} \\
&\quad + \frac{\partial}{\partial \tau_1} \nabla_1 \cdot \sum_{i=1}^M \mathbf{e}_i f_i^{\text{eq}} + \frac{1}{2} \nabla_1 \nabla_1 : \sum_{i=1}^M \mathbf{e}_i \mathbf{e}_i f_i^{\text{eq}} \\
&= \frac{\partial}{\partial \tau_2} \rho + \nabla_1 \cdot j^{(1)} + \frac{1}{2} \frac{\partial^2}{\partial \tau_1^2} \rho + \frac{\partial}{\partial \tau_1} \nabla_1 \cdot j^{(0)} + \frac{1}{2} \nabla_1 \nabla_1 : \Pi^{(0)} + \frac{1}{2} c_s^2 \nabla_1^2 \rho = 0 \quad (23)
\end{aligned}$$

where, c_s is defined as a sound speed, and

$$\Pi = \sum_{i=1}^M (\mathbf{e}_i \mathbf{e}_i - c_s^2 I) f_i \quad (24)$$

To eliminate second order derivative of time, insert Eq. (22) into Eq. (23), and then the following equation is obtained as:

$$\frac{\partial}{\partial \tau_2} \rho + \nabla_1 \cdot j^{(1)} + \frac{1}{2} \frac{\partial}{\partial \tau_1} \nabla_1 \cdot j^{(0)} + \frac{1}{2} \nabla_1 \nabla_1 : \Pi^{(0)} + \frac{1}{2} c_s^2 \nabla_1^2 \rho = 0 \quad (25)$$

In order to remove the second order time derivative of density, introduce a source term $F = \varepsilon F_i^{(1)}$ and put it into Eq. (14).

$$\sum_{i=1}^M \Omega_i \mathbf{e}_i = F \quad (26)$$

Due to the source term, a correction term should be added to momentum as:

$$j = \sum_{i=1}^M f_i \mathbf{e}_i = \sum_{i=1}^M f_i^{\text{eq}} \mathbf{e}_i - \frac{F}{2} \quad (27)$$

Extracting terms for $O(\varepsilon)$ from Eq. (26), then

$$\sum_{i=1}^M \mathbf{e}_i \Omega_i^{(1)} = \frac{\partial}{\partial \tau_1} \sum_{i=1}^M \mathbf{e}_i f_i^{\text{eq}} + \nabla_1 \cdot \sum_{i=1}^M \mathbf{e}_i \mathbf{e}_i f_i^{\text{eq}} = \frac{\partial}{\partial \tau_1} j^{(0)} + \nabla_1 \cdot \Pi^{(0)} + c_s^2 \nabla_1 \rho = F_i^{(1)} \quad (28)$$

Now, terms for $O(\varepsilon^2)$ in Eq. (25) can be evaluated using Eq.'s (27) and (28) as:

$$\frac{\partial}{\partial \tau_1} \nabla_1 \cdot j^{(0)} = \nabla_1 \cdot F_i^{(1)} - \nabla_1 \nabla_1 : \Pi^{(0)} - c_s^2 \nabla_1^2 \rho \quad (29)$$

$$\nabla_1 \cdot j^{(1)} = -\frac{1}{2} \nabla_1 \cdot F \quad (30)$$

Inserting above two equations into Eq. (25), finally the equation is obtained as:

$$\frac{\partial}{\partial \tau_2} \rho = 0 \quad (31)$$

This equation forms the mass conservation equation of NS equation eliminating τ_2 time scale from Eq. (21) as:

$$\frac{\partial}{\partial \tau_1} \rho + \nabla \cdot j = 0 \quad (32)$$

Extracting terms for $O(\varepsilon^2)$ from Eq. (26), then

$$\begin{aligned} \sum_{i=1}^M e_i \Omega_i^{(2)} &= \frac{\partial}{\partial \tau_1} \sum_{i=1}^M e_i f_i^{(1)} + \frac{\partial}{\partial \tau_2} \sum_{i=1}^M e_i f_i^{\text{eq}} \\ &\quad + \nabla_1 \cdot \sum_{i=1}^M e_i e_i f_i^{(1)} + \frac{1}{2} \frac{\partial^2}{\partial \tau_1^2} \sum_{i=1}^M e_i f_i^{\text{eq}} + \frac{\partial}{\partial \tau_1} \nabla_1 \cdot \sum_{i=1}^M e_i e_i f_i^{\text{eq}} + \frac{1}{2} \nabla_1 \nabla_1 : R_i^{(0)} \\ &= \frac{\partial}{\partial \tau_1} j^{(1)} + \frac{\partial}{\partial \tau_2} j^{(0)} + \nabla_1 \cdot \Pi^{(1)} + \frac{1}{2} \frac{\partial^2}{\partial \tau_1^2} j^{(0)} + \frac{\partial}{\partial \tau_1} \nabla_1 \cdot \Pi^{(0)} + c_s^2 \frac{\partial}{\partial \tau_1} \nabla_1^2 \rho + \frac{1}{2} \nabla_1 \nabla_1 : R_i^{(0)} = 0 \end{aligned} \quad (33)$$

where the tensor R_i is nonlinear deviation term [6].

Using Eq. (28), the second order time derivative of zeroth-order of momentum can be written as:

$$\frac{\partial^2}{\partial \tau_1^2} j^{(0)} = \frac{\partial}{\partial \tau_1} \left(F_i^{(1)} - \nabla_1 \cdot \Pi^{(0)} - c_s^2 \nabla_1 \rho \right) \quad (34)$$

Inserting the equation into Eq. (33) and using Eq. (30), and then Eq. (33) can be simplified as:

$$\frac{\partial}{\partial \tau_2} j^{(0)} + \nabla_1 \cdot \Pi^{(1)} + \frac{1}{2} \frac{\partial}{\partial \tau_1} \nabla_1 \cdot \Pi^{(0)} + \frac{1}{2} c_s^2 \frac{\partial}{\partial \tau_1} \nabla_1^2 \rho + \frac{1}{2} \nabla_1 \nabla_1 : R_i^{(0)} = 0 \quad (35)$$

To make momentum conservation equations, solve Eq. (26) combining Eq.'s (28) and (35):

$$\frac{\partial}{\partial \tau} j^{(0)} + \nabla \cdot \left(\Pi + c_s^2 \rho I + \frac{\varepsilon}{2} \left(\frac{\partial}{\partial \tau_1} (\Pi^{(0)} + c_s^2 \rho I) + \nabla_1 \cdot R^{(0)} \right) \right) = F \quad (36)$$

The terms of Eq. (36) can be matched with macroscopic momentum equations such as: $\Pi^{(0)}$ is identified with $\rho u u$, $c_s^2 \rho I$ with $p I$ by ideal gas law, and the remaining $O(\varepsilon^2)$ terms with $-\tau$. For example, let's take two-dimensional lattice with nine velocity vectors which has origin $e_i(0,0)$ in Figure 2, which is easy to understand mathematical manipulating instead of three-dimensional model. Chen et al. [7] formulated the general form of equilibrium distribution function, which has the error $O(u^2)$, as:

$$f_i^{\text{eq}} = \rho [a + b e_i \cdot u + c (e_i \cdot u)^2 + d u^2] \quad (36)$$

where, a, b, c, and, d are unknown constants with the assumption of small Mach numbers. Satisfying Eq.'s (11) and (12), the unknown constants can be analytically found as [8]:

$$f_i^{\text{eq}} = \rho \omega_i \left[1 + 3\mathbf{e}_i \cdot \mathbf{u} + \frac{9}{2} (\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2} u^2 \right] \quad (37)$$

where, $\omega_1=4/9$, $\omega_i=1/9$ ($i = 2,4,6,\text{and } 8$),and $\omega_i=1/36$ ($i = 3,5,7,\text{and } 9$) which are determined to accomplish isotropy of the fourth-order tensor of velocities and Galilean invariance [9].

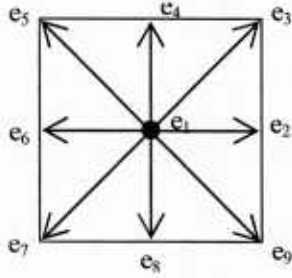


Figure 2. A two-dimensional lattice with nine velocity vectors

Inserting the above equation into Eq. (36) and solve it, and finally the NS momentum equation are obtained as:

$$\frac{\partial \mathbf{u}}{\partial \tau} + \nabla \cdot \mathbf{u} \mathbf{u} = \frac{1}{\rho} [-\nabla p + \nu \nabla \cdot (\nabla \rho \mathbf{u} + \nabla \rho \mathbf{u})] \quad (38)$$

where, $p = \rho/3$ is defined as the pressure, which provides a sound speed, $c_s = 1/\sqrt{3}$ by ideal gas law, and $\nu = (2\tau - 1)/6$ is denoted as the kinematic viscosity.

Regularized Lattice Boltzmann

Before going to the details of the regularization procedure [10] of LB, let's recall original LB equation, Eq. (2)

$$f_i(\mathbf{x} + \mathbf{e}_i, t + 1) = f_i(\mathbf{x}, t) + \Omega_i(f_i(\mathbf{x}, t)), \quad (i = 1, \dots, N)$$

The dynamics of particles in the LB equation can be split into two steps: 1) collision step, 2) stream step (propagation step). The former calculates the new outgoing particle parameters from

incoming ones by the relation $f_i^{\text{out}} = f_i^{\text{in}} + \Omega_i(f_i(x, t))$, and the latter propagate incoming particles along the lattice velocity e_i by the relation $f_i^{\text{in}}(x + e_i, t + 1) = f_i^{\text{out}}$. Let's define $\bar{\Pi}$ as momentum flux tensor as:

$$\bar{\Pi}_{\alpha\beta} = \sum_{i=1}^M e_{i\alpha} e_{i\beta} f_i \quad (39)$$

where, $e_{i\alpha}$ is the component of the velocity vector e_i in α -coordinate direction. Using the BGK equation, Eq. (16) for modeling dynamics of particles in LB equation, the distribute functions f_i can simply be defined as:

$$f_i = f_i^{\text{eq}} + f_i^{\text{neq}} \quad (40)$$

and then the momentum flux of nonequilibrium part of the distribution functions is calculated as:

$$\bar{\Pi}_{\alpha\beta}^{\text{neq}} = \bar{\Pi}_{\alpha\beta} - \sum_{i=1}^M e_{i\alpha} e_{i\beta} f_i^{\text{eq}} \quad (41)$$

By cancelling higher order contributions from CE expansion to BGK dynamics to comply with their $O(\epsilon)$ hydrodynamic values, the nonequilibrium part of the distribution function can be expressed as:

$$f_i^{\text{neq}} \approx f_i^{(1)} = -\frac{\omega_i \tau}{c_s^2} (e_{i\alpha} e_{i\beta} - c_s^2 I) \nabla_\alpha \rho u_\alpha \quad (42)$$

, and then the momentum flux of nonequilibrium part is calculated as:

$$\bar{\Pi}_{\alpha\beta}^{\text{neq}} \approx \sum_{i=1}^M e_{i\alpha} e_{i\beta} f_i^{(1)} = -\tau c_s^2 (\nabla_\alpha \rho u_\alpha + \nabla_\beta \rho u_\beta) \quad (43)$$

Combining Eq's (42) and (43), the regularized nonequilibrium part of the distribution function can be simplified as:

$$f_i^{(1)} = \frac{\omega_i}{2c_s^2} (e_{i\alpha} e_{i\beta} - c_s^2 I) \bar{\Pi}_{\alpha\beta}^{\text{neq}} \quad (44)$$

This equation provides regularized distribution functions, $f_i^{\text{reg}} = f_i^{\text{eq}}(\rho, u) + f_i^{(1)}$ that possess the required symmetries for lattice BGK model.

Moving Boundary in Lattice Boltzmann Method

To handle moving boundaries in LB method, which is required to analyze the interaction between a representative model for nanotube and polymer, the method developed by Lallemand and Luo [11] has been employed in OpenLB code. The brief description of the method is described and the implementation procedure in OpenLB is explained.

Moving Boundary Algorithm

The moving boundary algorithm proposed by Lallemand and Luo [11] is based on the quadratic interpolation and bounce-back scheme on curved boundaries. The bounce-back scheme is implemented with the assumption that the interface boundary between two different materials (ex.: solid and fluid) is located on the half-way in a cell. The challenge is how to solve moving interface not located on the half-way in Figure 3. A particle, x is located on r_i in the Figure, is considered in one direction along one velocity vector in Figure 2. Let's define q , the distance between the interface boundary and a closest fluid node, which is normalized to 1. In addition, consider a moving wall located at an arbitrary position r_w between two nodes, r_j and r_s , where r_j is located on the fluid region and r_s on the moving solid.

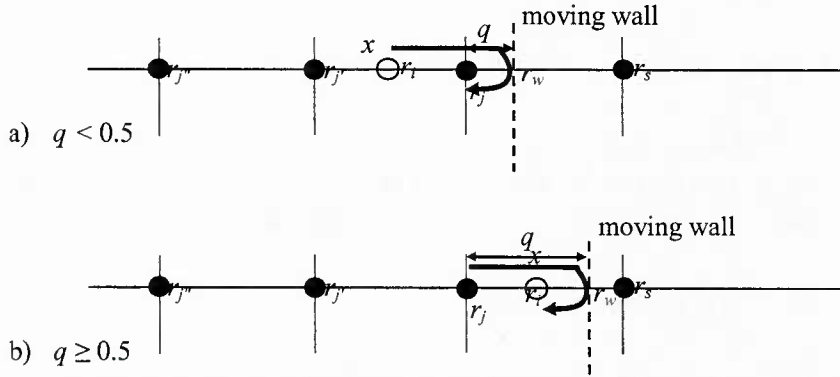


Figure 3. Two cases for moving interface boundary in the lattice used in LB method

For the case of $q < 0.5$, the distribution function, $f_i(r_j, t)$, where the under bar of i , one velocity vector, denotes inverse direction to the vector, can be interpolated before propagation and collided. The distribution function can be written as [11]:

$$f_{\bar{i}}(r_j, t) = q(1 + 2q)\tilde{f}_{\bar{i}}(r_j, t) + (1 - 4q^2)\tilde{f}_{\bar{i}}(r_{j'}, t) - q(1 - 2q)\tilde{f}_{\bar{i}}(r_{j''}, t) + 3\omega_i(\mathbf{e}_i \cdot \mathbf{u}_w) \quad (45)$$

where $\tilde{f}_{\bar{i}}$ is the distribution from the previous time (before propagation), and \mathbf{u}_w is the wall velocity.

For the case of $q \geq 0.5$, the distribution function, $f_{\bar{i}}(r_j, t)$, can be interpolated after propagation and collided. The distribution function can be written as [11]:

$$f_{\bar{i}}(r_j, t) = \frac{1}{q(1+2q)}\tilde{f}_{\bar{i}}(r_j, t) - \frac{(1-2q)}{q}\tilde{f}_{\bar{i}}(r_{j'}, t) - \frac{(2q-1)}{(2q+1)}\tilde{f}_{\bar{i}}(r_{j''}, t) + \frac{3\omega_i}{q(1+2q)}(\mathbf{e}_i \cdot \mathbf{u}_w) \quad (46)$$

The last term, $\omega_i(\mathbf{e}_i \cdot \mathbf{u}_w)$ in Eq's (45) and (46), describes the momentum due to interaction between fluid and solid introduced from the mass and momentum conservation.

Since propagation in time is the same as movement from a location to next to the location in space, the Eq's (45) and (46) can be also written as:

For the case of $q < 0.5$,

$$f_i(r_j, t) = q(1 + 2q)f_i(r_j + e_i \Delta t, t) + (1 - 4q^2)f_i(r_j, t) - q(1 - 2q)f_i(r_j - e_i \Delta t, t) + 3\omega_i(e_i \cdot u_w) \quad (47)$$

For the case of $q \geq 0.5$,

$$f_i(r_j, t) = \frac{1}{q(1+2q)}f_i(r + e_i \Delta t, t) - \frac{(1-2q)}{q}f_i(r_j - e_i \Delta t, t) - \frac{(2q-1)}{(2q+1)}f_i(r_j - 2e_i \Delta t, t) + \frac{3\omega_i}{q(1+2q)}(e_i \cdot u_w) \quad (48)$$

Implementation of Moving Boundary Algorithm in OpenLB

To implement the moving boundary algorithm in OpenLB, the hierarchical data structure of OpenLB has been analyzed to touch the distribution function and get momentum introduced from fluid structure interaction. For the distance q , bisection method has been employed.

General setup for stationary fluid dynamics in OpenLB

To simulate stationary fluid dynamics in OpenLB, the following setup is required.

- Define flow parameters involving Reynolds number and lattice velocity.
- Define boundary and initial condition
- Define termination criteria.
- Define flow field with object region in the lattice using 0 and 1 representing flow and object, respectively
- Choose the model for flow dynamics involving "BGKdynamics", "MomentumExchangeBounceBack", "ExternalMomentBGKdynamics", and etc.
- Initialize distribution function and collide and stream until the given termination criteria

The modification of the general procedure is required to implement the moving boundary algorithm.

Modification of general setup for moving boundary algorithm in OpenLB

To simulate moving boundary algorithm in OpenLB, the following setup is required.

- Define flow parameters involving Reynolds number and lattice velocity.
- Define boundary and initial condition
- Define termination criteria.

- d. Define initially and update flow field with object region in the lattice using 0 and 1 representing flow and moving object, respectively
- e. Choose initially and update the “MomentumExchangeBounceBack” model for moving object and “BGKdynamics” for flow region.
- f. Initialize distribution function and collide and stream initially once.
- g. Find the distance q using bisection method and update distribution function using Eq’s (47) and (48)

The modification of the general procedure is required to implement the moving boundary algorithm. In the several model of dynamics of distribution function in OpenLB, a “MomentumExchangeBounceBack” class is chosen for defining solid regions to get momentum on the interface boundaries between fluid and moving solid objects.

Distribution function update in OpenLB

The final step in the previous section has been implemented modifying codes in OpenLB. The bisection method [12] was used to obtain the distance q . Flow direction information was extracted in the “dynamic” class in OpenLB to find neighboring distribution function and get the value in the function along the flow direction. This procedure was performed using follow steps.

- a. Find interface cell between fluid and moving solid object using the dynamic model information.
- b. Get the flow directions in the “dynamic” class in the found interface cells
- c. Get the distances using the bisection method.
- d. Get the index of neighboring distribution function and needed values of the function
- e. Update distribution function on moving boundary using Eq’s (47) and (48) with the obtained values in previous steps.
- f.

Two dimensional flow past a moving cylinder

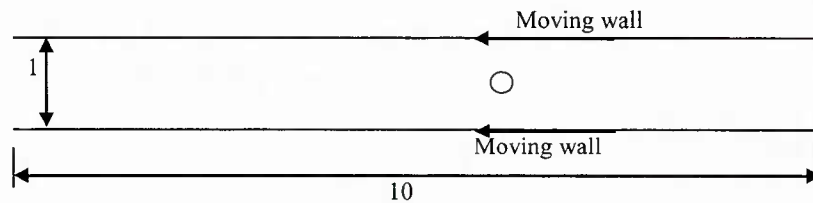
Two dimensional flow past a moving cylinder eccentrically located in a channel has been simulated to validate the moving boundary algorithm using the two frames of reference. One is the moving frame with a fixed cylinder. The other is the fixed frame with a moving cylinder. The boundary and initial conditions listed in Table 1 were setup to make the relative motion between the cylinder and the flow in the channel the same in either frame.

Table 1. Different setup conditions for the flow past a moving cylinder

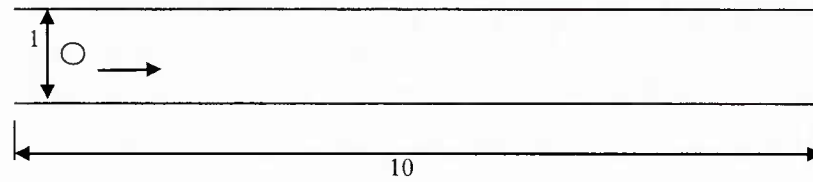
	Moving Frame	Fixed Frame
Inlet	Constant Pressure	Constant Pressure
outlet	Outflow	Outflow

Side Wall	Moving and no-slip	No-slip and no-movement
Cylinder	Fixed	Moving

The Reynolds number for this simulation is taken as 200. The height and length dimension of the channel is 1 and 10 respectively. The initial velocity is 0.0 in the moving frame. The side walls are moving with -0.04 velocity. In the fixed frame, the cylinder is moving with 0.04 velocity. All movement is only along x -direction. The cylinder has 0.12 radius. It is located on (6.6, 0.54) in the moving frame and (0.6, 0.54) in the fixed frame, as shown in Figure 4. The contour plots of velocity in x -direction in both frames are shown in Figure 5. The velocity in x -direction in moving frame is shifted by 0.04 to compare the results with those in fixed frame. The results from fixed frame showed lots of fluctuation, which may be caused by taking averaged values of the nearest nodes in the extrapolation of distribution function from solid to fluid region due to the moving boundaries.

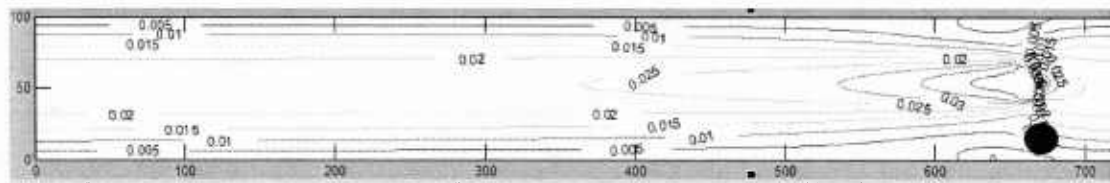


a) A cylinder in moving frame

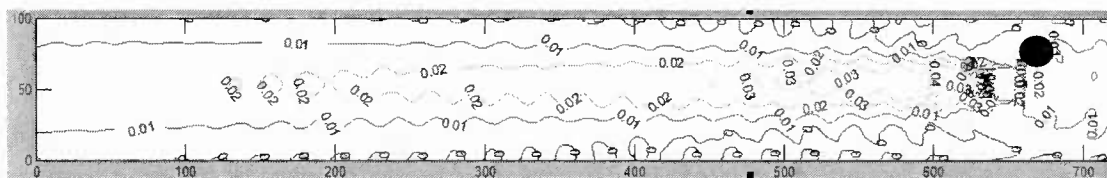


b) A moving cylinder in fixed frame

Figure 4. A cylinder in a channel with different frames



a) A cylinder in moving frame



b) A moving cylinder in fixed frame

Figure 5. Contour plot of flow velocity in x direction

References

1. He X., and Luo L-S., "A priori derivation of the lattice Boltzmann equation," *Physical Review E*, Vol. 55, 1997, pp. 6333–6336
2. Frisch U., d'Humieres D., Hasslacher B., Lallemand P., Pomeau Y., and Rivet J-P., "Lattice gas hydrodynamics in two and three dimensions," *Complex Syst.*, Vol.1, 1987, pp. 649–707
3. Higuera F.J., and Jimenez J., "Boltzmann approach to lattice gas simulations," *Europhysics Letter*, Vol. 9, 1989, pp.663–668
4. Bhatnagar P.L., Gross E.P., and Krook M., "A model for collision processes in gases. I: small amplitude processes in charged and neutral one-component system," *Phys. Rev.*, Vol.94, 1954, pp.511–525
5. Frisch U., Hasslacher B., and Pomeau Y., "Lattice-gas automata for the Navier-Stokes equations," *Physical Review Letter*, Vol.56, 1986, pp. 1505–1508
6. Qian Y.H., and Orszag S.A., "Lattice BGK models for the Navier-Stokes equation: nonlinear deviation in compressible regimes," *Europhys. Lett.*, Vol. 21, 1993, pp. 255–259
7. Chen H., Chen S., and Matthaeus W.H., "Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method," *Phys. Rev. A*, Vol.45, 1992, pp. R5339–42
8. Qian Y.H., d'Humieres D., and Lallemand P., "Lattice BGK models for Navier-Stokes equation," *Europhys. Lett.*, Vol. 17, 1992, pp. 479–484
9. Qian Y.H., "Lattice gas and lattice kinetic theory applied to the Navier-Stokes equations," PhD thesis. Universit'e Pierre et Marie Curie, Paris, 1990
10. Latt J., and Chopard B., "Lattice Boltzmann method with regularized pre-collision distribution functions," *Math. Comput. Simulat.*, Vol. 72, 2006, pp. 165-168

11. Lallemand P. and Luo L-S., "Lattice Boltzmann method for moving boundaries," *Journal of Computational Physics*, Vol. 184 , No. 2, Jan. 2003, pp. 406 – 421
12. Burden R. and Faires J.D. , "2.1 The Bisection Algorithm," *Numerical Analysis* (3rd ed.), PWS Publishers, 1985

B-1 APPENDIX-I

Preliminary Results

The main sections discussed the theoretical and mathematical formulations related to the Lattice Boltzmann simulations and their relation to the macroscopic flow variables. Preliminary computational investigations that were performed in the project efforts to understand the usage and behavior of Lattice Boltzmann method for flow problems are presented next. Classic flow configurations in 2D and 3D geometrical configurations are the preliminary test simulations investigated.

Lattice Boltzmann Simulation

OpenLB Test Cases

A Lattice Boltzmann simulation solver, OpenLB, has been downloaded to simulate two cases for multi-scale simulation to verify whether the open source code works in the UAB Linux-cluster system. The first case is a flow around a 2D cylinder inside a channel, which produces a von Karman vortex street. The Reynolds number for this simulation is taken as 400 and the geometry used for this simulation is illustrated in Figure 1. The captured pictures of the flow development from the simulation are shown in Figure 2. The simulation took about 23 minutes of CPU time.

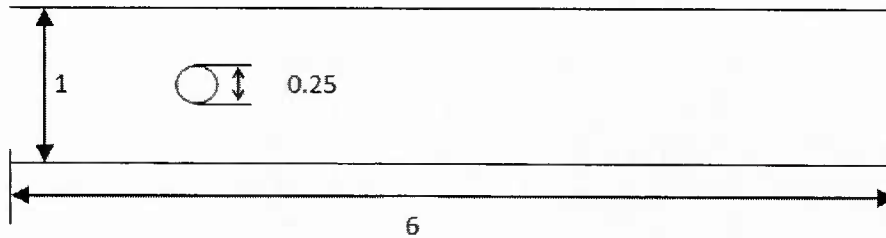


Figure 1: Geometry used for flow past a cylinder using Lattice Boltzmann equation

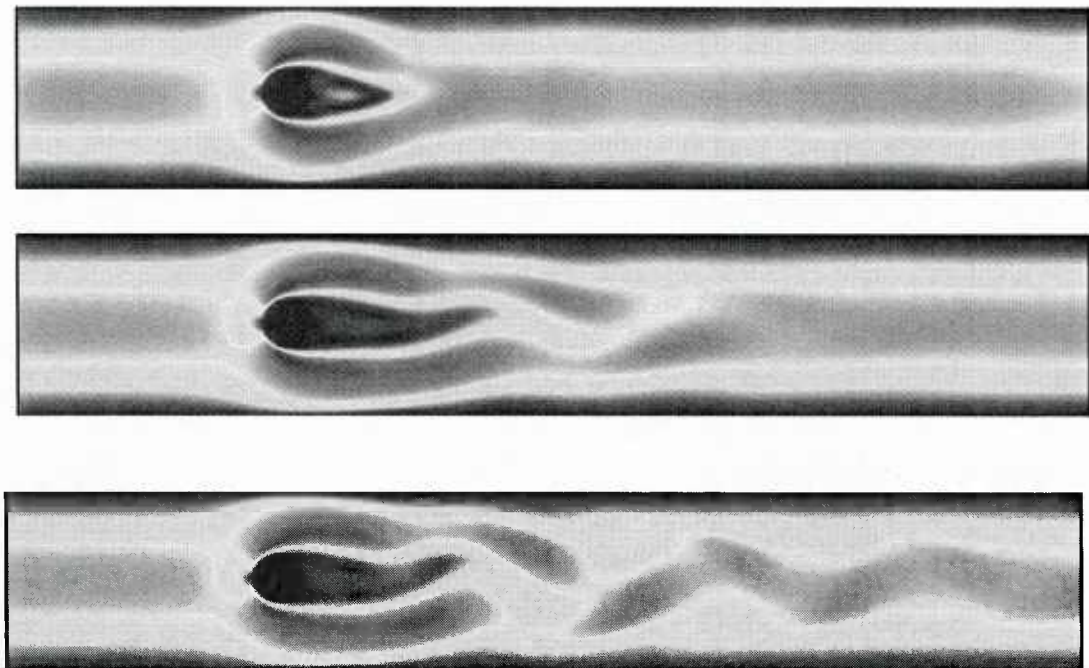


Figure 2: Von Karman Vortex Street past a Cylinder by Lattice Boltzmann Equation

The second test case is a driven cavity flow. The geometry for this problem is a square cavity with upper surface moving at a constant speed. The Reynolds number for this problem is set to 100. The captured pictures of the animation of the energy contour from the simulation are shown in Figure 3.

The third test case is a flow past a 3D cylinder has been simulated by LB using BGK dynamics and regularized LB methods. The dimension of geometry is illustrated in Figure 6. The mesh was constructed using a template “CylinderShaeDomain3D()” in OpenLB, which is modified based on a built-in template “CylinderShaeDomain2D()”. The resolution was set to 101 nodes along each direction. The Reynolds number for this simulation is taken as 400. The velocities at the top, bottom, and side walls were set to zero, the velocity at the outlet was extrapolated from the inside, and the velocity at the inlet was set based on the Poiseuille profile.

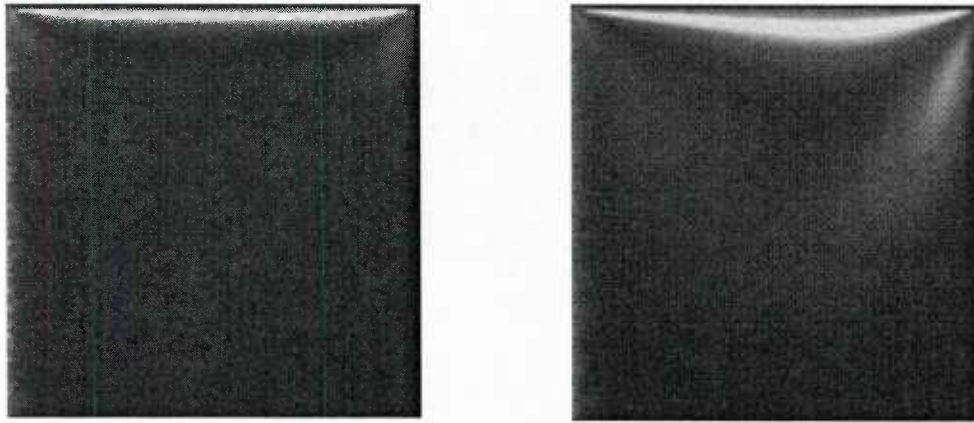


Figure 3: Energy contour of driven cavity flow by Lattice Boltzmann equation

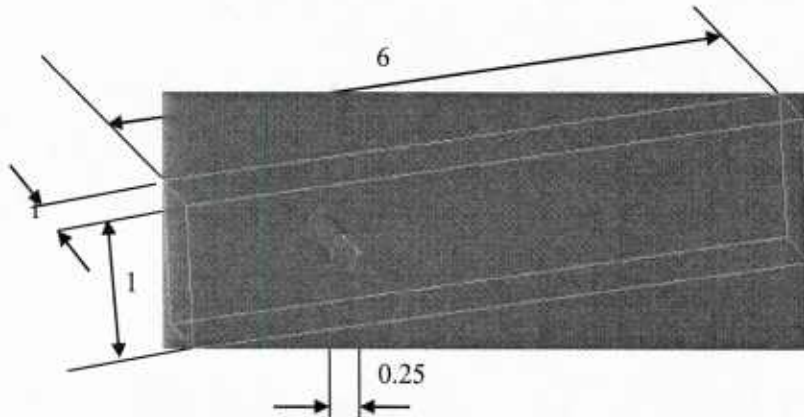
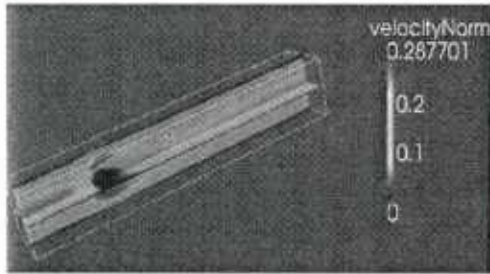
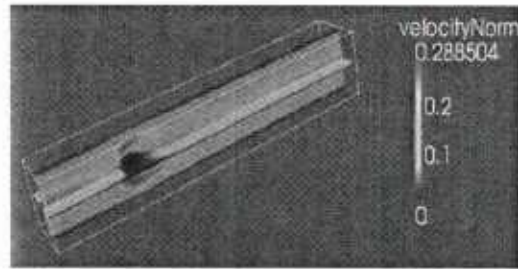


Figure 4: Dimension of cylinder and far-field domain for 3D cylinder simulation

The velocity distribution, velocity vectors, and streamlines predicted by the LB method using BGK dynamics and regularized LB method are shown in Figures 5-7. The velocity vectors and the traces are colored based on the velocity magnitude (blue color represents lowest velocity and red color represents highest velocity). In Figure 5, the maximum velocity from Regularized LBE shows good agreement with LBE using BGK with 0.28% error and the distributions at two cross sections are almost the same. Each simulation took about 29 days using 16 processors. The good agreement between both methods as shown in Figures 5-6 indicate that the Regularized LB method can be substituted for LB method using BGK dynamics for easily coupling between LB and NS solvers.

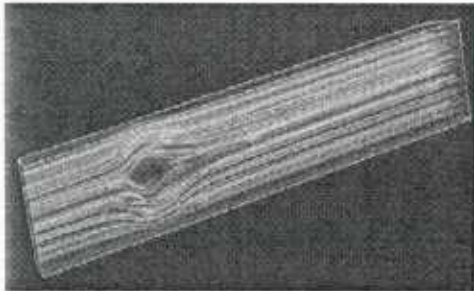


a) Velocity from LBE using BGK

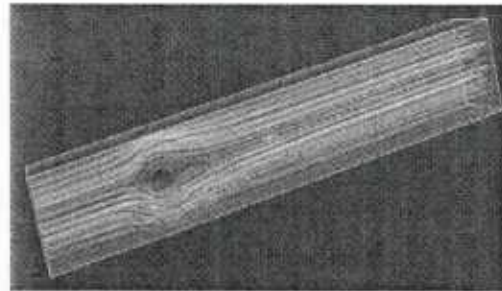


b) Velocity from Regularized LBE

Figure 5: Velocity distributions from LBE using BGK and Regularized LBE

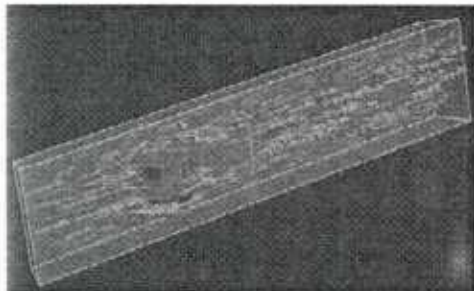


a) Particle traces from LBE using BGK

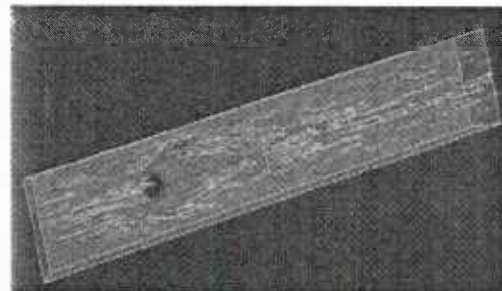


b) Particle traces from Regularized LBE

Figure 6: Particle traces from LBE using BGK and Regularized LBE



a) Velocity vectors from LBE using BGK



b) Velocity vectors from Regularized LBE

Figure 7: Velocity vectors from LBE using BGK and Regularized LBE

Navier-Stokes Simulation

We have simulated a laminar flow over a cylinder using the continuum approach via a UAB in-house finite volume code, HYB3D. The Reynolds number for this simulation is taken as 335.51. A cross-sectional view of the mesh and pressure distribution on the surface of the

cylinder are shown in Figure 8. The pressure distribution, velocity vectors, and streamlines predicted by the Navier-Stokes simulation are given in Figures 9-11. The velocity vectors and the streamlines are colored based on the velocity magnitude (blue color represents lowest velocity and red color represents highest velocity). In all these plots the cylinder is colored based on the pressure distribution.

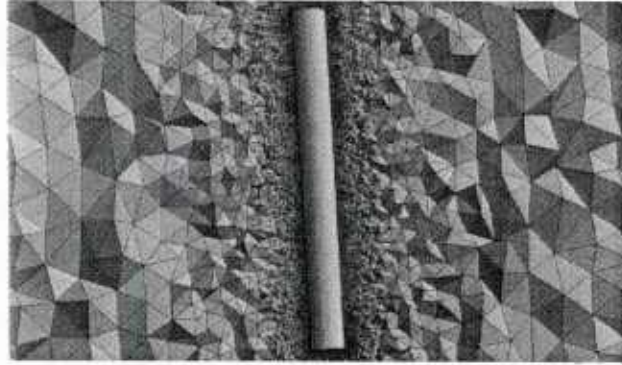


Figure 8: Cross-sectional view of the mesh for HYB3D

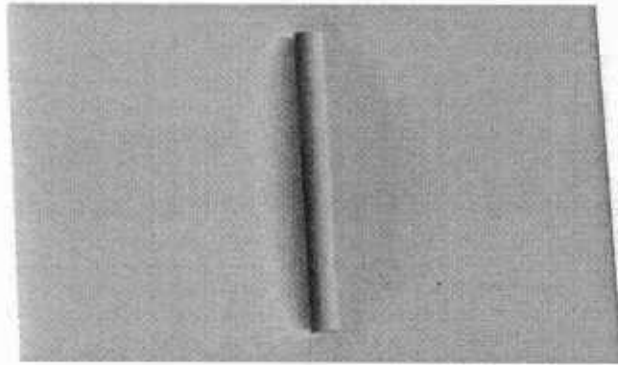
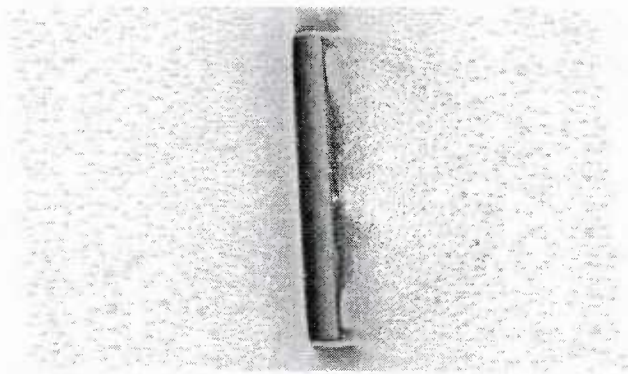
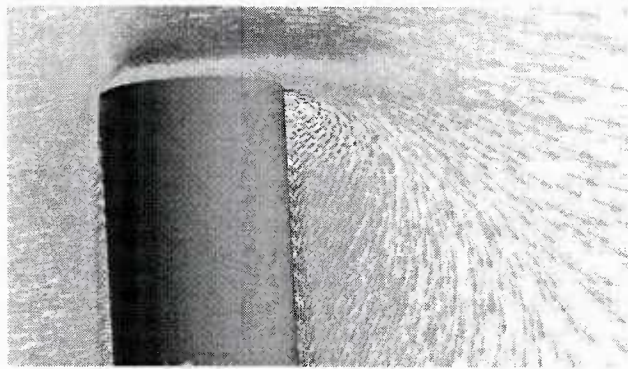


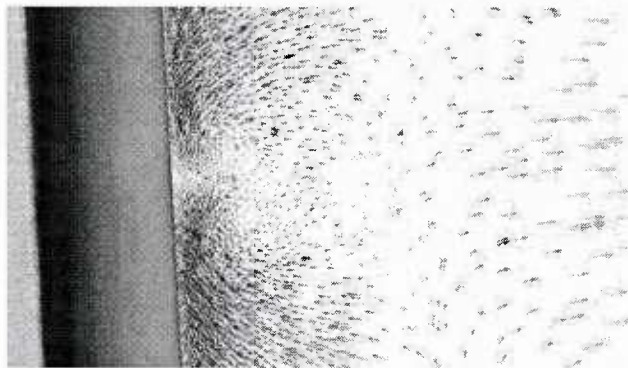
Figure 9: Pressure distribution by HYB3D



(a) Overall view

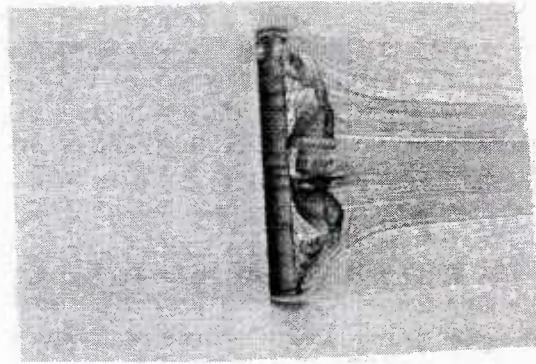


(b) Upper end of the cylinder

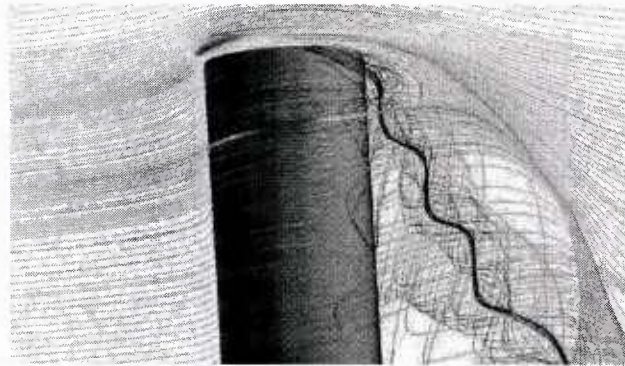


(c) Middle section

Figure 10: Velocity vectors by HYB3D



(a) Overall view



(b) Near the end

Figure 11: Streamlines by HYB3D

B-1 APPENDIX-II

(Project Results)

Poiseuille flow simulation for validation of OpenLB and HYB3D codes

Before the validation of coupling scheme between two solvers, the validation of each solver should be carried out. Therefore, the validation of each solver has been conducted by Poiseuille flow through a square duct. The results show good agreement with analytical solution. This procedure provides the important information involving unit coincidence of parameters requiring for time and space synchronization, boundary conditions, initial conditions, and required input parameters for each solver.

Poiseuille flow through square duct

Poiseuille flow through square duct has been analyzed for the validation of two solvers, OpenLB and HYB3D. The Reynolds number for this simulation is taken as 100. The air dynamic viscosity is 1.5×10^{-5} kg/m*s. The height and length of square duct is 0.01m, and 0.05m, respectively. The initial velocity is 0.15 m/s at the inlet. The analytical equation for this flow can be written as

$$u = \frac{16a^2}{\mu \pi^3} \left(-\frac{dp}{dx} \right) \sum_{i=1,3,5,\dots}^{\infty} (-1)^{(i-1)/2} \left[1 - \frac{\cosh(i\pi y/2a)}{\cosh(i\pi/2)} \right] \frac{\cos(i\pi y/2a)}{i^3} \quad (1)$$

where, $-a \leq y \leq a$, and a is defined as one half of height.

Computational setup for Poiseuille flow

Computational setups for the flow through square duct have been established for two solvers. Geometrical information for the square duct and physical condition for the flow was fitted into computational parameters as listed in Table1. The required boundary and initial condition and the others to get results for each solver are listed in Table 2. The different mesh densities were employed to check the sensitivity of the mesh density to the computational results.

Table 1. Fitted conditions and geometry information for Poiseuille flow simulation

	Physical value	LBE	HYB3D
Re	100	100	100
Initial Velocity	0.15 m/s	1	1
dynamic viscosity	1.5×10^{-5} kg/m*s	N/A	N/A

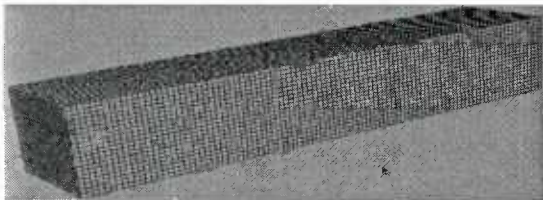
Height of square duct	0.01 m	1	1
Length of square duct	0.05 m	5	5
Time	N/A	Depends on mesh	Depends on mesh

Table 2. The boundary conditions and mesh quality for Poisseulle flow simulation

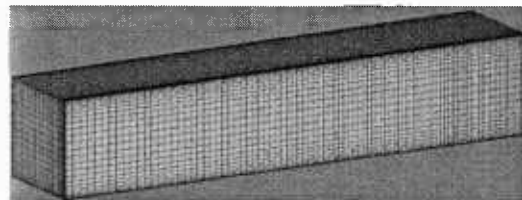
	LBE	HYB3D
Inlet	Inflow (velocity)	Inflow (velocity)
outlet	outflow	Outflow (pressure)
Side Wall	No-slip	No-slip
Mesh quality	20, 60, 80 (Height)	30, 50, 60 (Height)
Biasing for viscous layer	No	Yes
Parallel Processing	No	Yes

Mesh generation

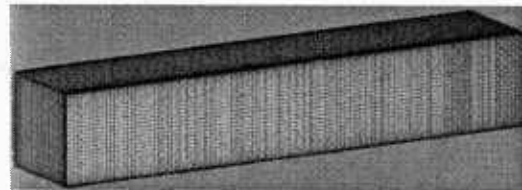
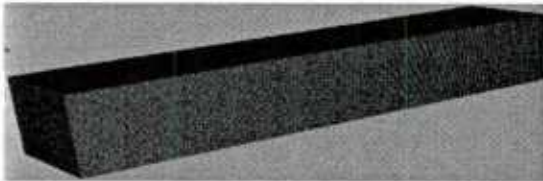
The flow domain through squared duct is automatically discretized to uniform hexahedral elements by OpenLB and elements in different sizes were generated manually for HYB3D using HyperMesh3D. Three different meshes for each solver have been generated and the total numbers of elements and nodes used in the meshes are described in Figure 1.



40,000 elements, 44,541 nodes



90,000 elements, 97,601 nodes



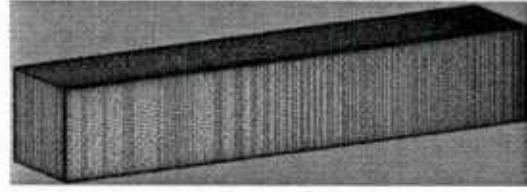
1,080,000 elements, 1,120,021 nodes



2,560,000 elements, 2,630,961 nodes

a) OpenLB

500,000 elements, 522,801 nodes



1,440,000 elements, 1,492,121 nodes

b) HYB3D

Figure 1. Three different meshes used in OpenLB and HYB3D for Poisseulle flow simulation

Velocity profile of Poisseulle flow through square duct

Velocity distribution and contour lines at the cross section in the middle of duct along length direction (x coordinate) are shown in Figure 2. For these results, the convergence history for both solvers is shown in Figure 3. OpenLB requires maximum 3,000 iterations for getting convergent solution, while HYB3D requires maximum 1,500 iterations. Comparing three results from OpenLB, the result obtained from the simulation with 40,000 elements show different velocity profile from the others. However, the results from HYB3D show good agreement each other. This indicates that the mesh with 1,080,000 elements for OpenLB and the mesh with 500,000 elements for HYB3D are good enough to get accurate results. The velocity contours and contour lines at the different cross sections along height direction (y coordinate) are shown in Figure 4. The results show that the sensitivity of the mesh density to velocity profile has similar trends as the velocity profile does along length direction. Figure 5 shows the velocity on the center line along length direction for both solvers. The plot in Figure 5 shows clearly the sensitivity of mesh density to the results.

To validate the results from two solvers with analytical solution using Eq. (49), the root mean square (RMS) errors on the cross section at $x = 4.9$ along height direction are calculated and shown in Figure 6. The surface plot and contour lines of the errors are shown in the Figure. The RMS errors are not small enough to show good agreement with analytical solution. The nonlinear velocity in Figure 3 indicates the 5 length is not enough to remove the entrance effect, so that the simulation has been implemented for the other lengths such as 10 and 20 lengths. The results with 10 length show the length is still not enough to get rid of the entrance effect as shown in Figure 7. The results with 20 lengths provide acceptable RMS errors as shown in Figure 8.

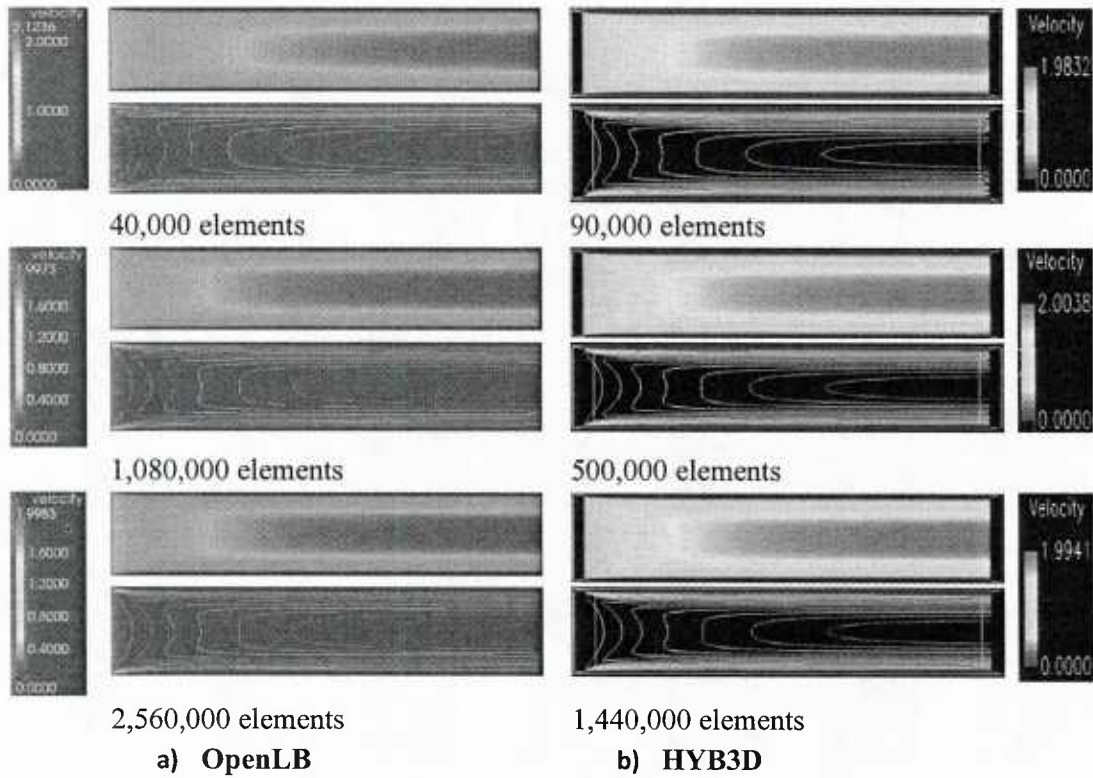


Figure 2. Velocity contours and contour lines at the cross section in the middle of duct along length direction of OpenLB and HYB3D codes

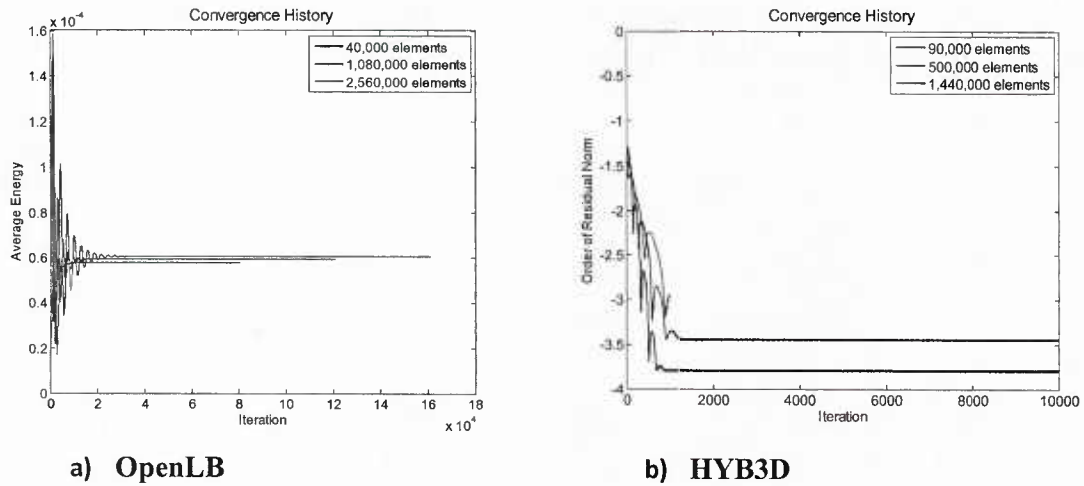


Figure 3. Convergence history for all meshes used in OpenLB and HYB3D

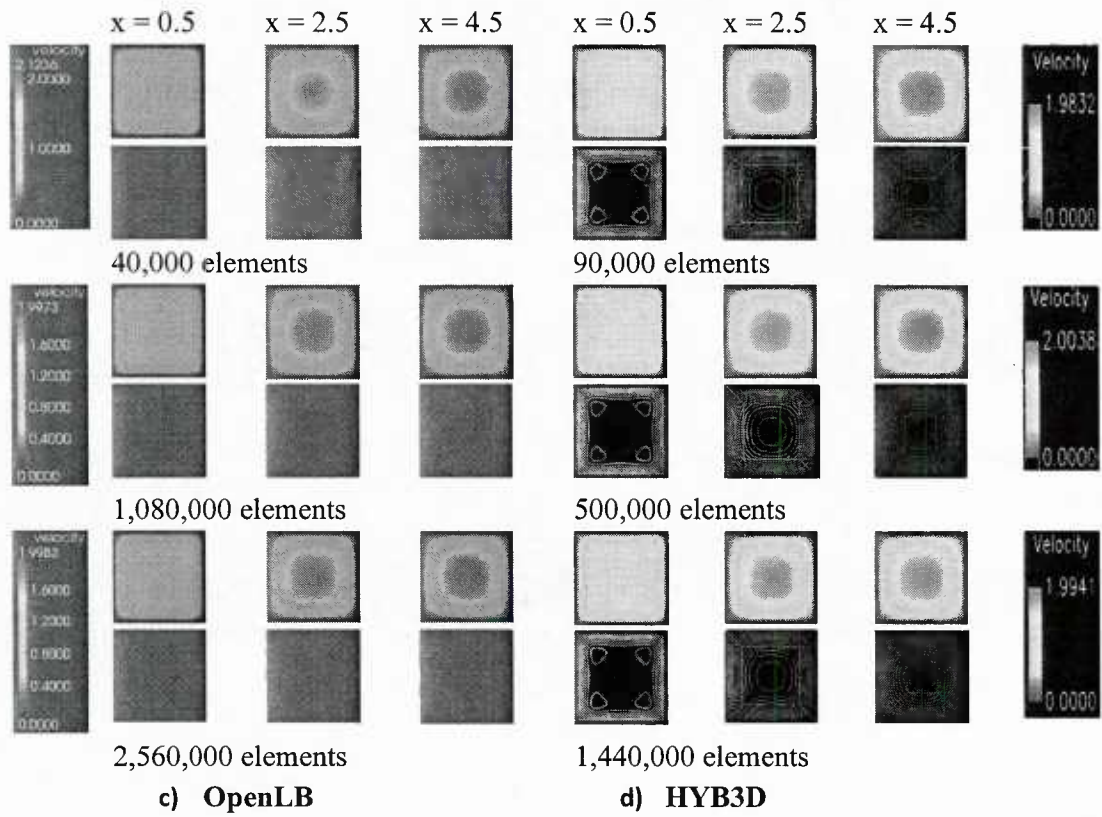


Figure 4. Velocity contours and contour lines at the cross sections ($x = 0.5$, 2.5 , and 4.5) along height direction of OpenLB and HYB3D codes

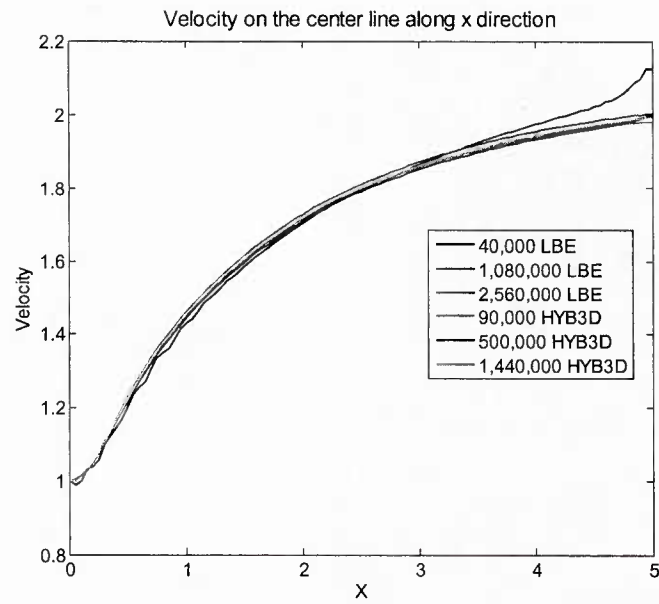


Figure 5. Comparison of velocities from OpenLB and HYB3D codes on the center line along length direction

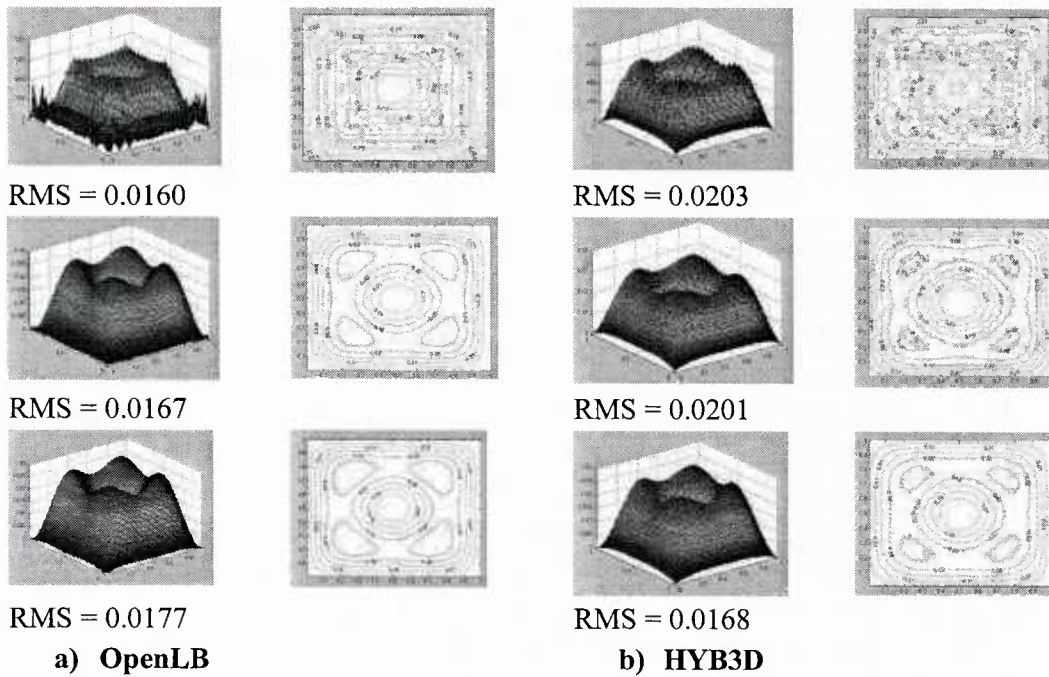


Figure 6. RMS errors at the cross section, $x = 4.9$ along height direction of OpenLB and HYB3D codes

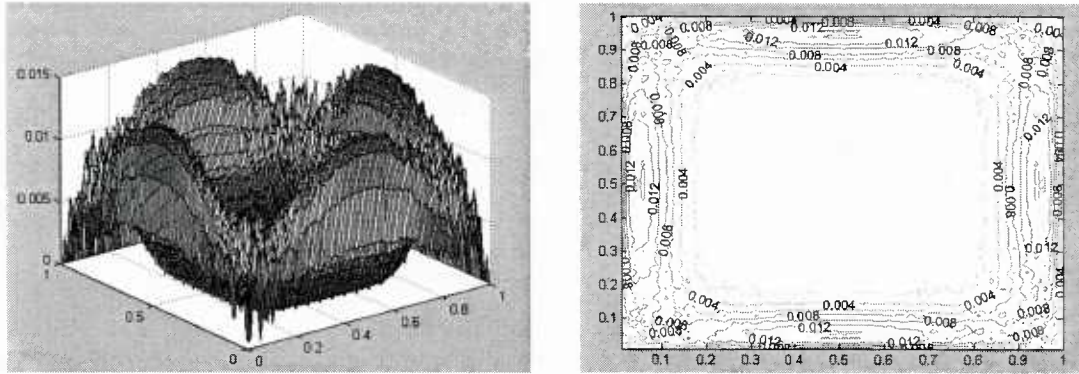


Figure 7. RMS errors at the cross section, $x = 9.95$ along height direction of OpenLB

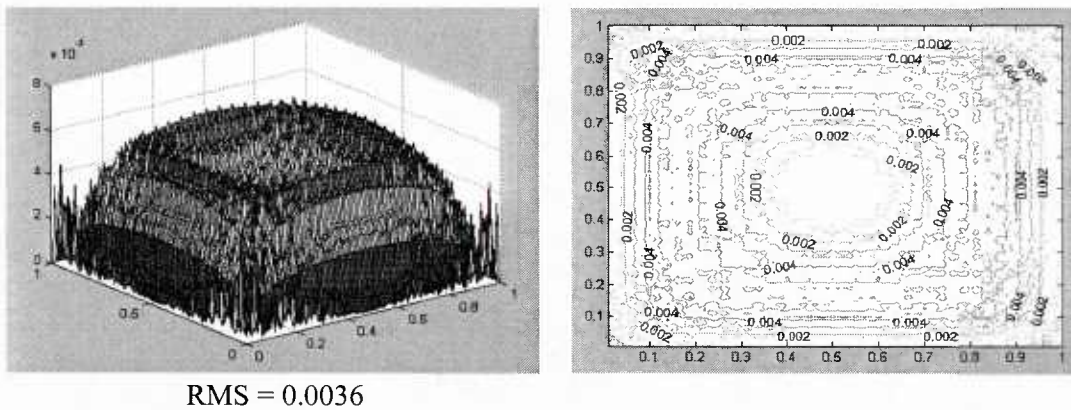


Figure 8. RMS errors at the cross section, $x = 18$ along height direction of OpenLB

Coupling OpenLB with HYB3D

Mesh generation

The coupling of OpenLB and HYB3D is carried out using the geometry sketched in Figure 9. The inside cubic region shown in yellow is solved using OpenLB and the rest of the domain is solved using the continuum approach. The spatial discretization of the OpenLB is carried out automatically using the built-in functions. The mesh for the continuum domain is generated using HyperMesh3D. The surface nodes from the OpenLB are taken as the interface nodes for the continuum domain as shown in Figure 10. From the surface mesh, 25 boundary layers with a growth rate of 1.2 and an initial thickness $1.0e-05$ are generated for viscous layer on the walls. The cross section view of the generated tetrahedral volume mesh in the middle of square duct is shown in Figure 11. The total number of tetrahedral elements for the continuum domain is 1,327,000. A zoomed-in view of the interface region is shown in Figure 12.

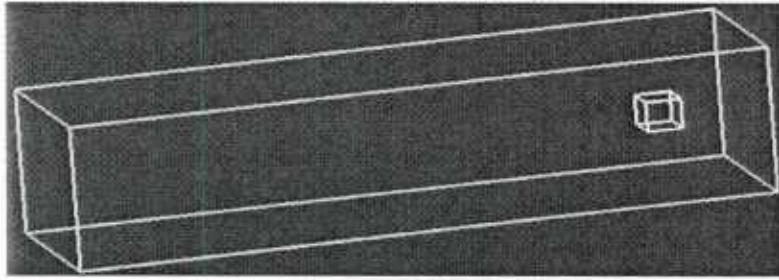


Figure 9. The location of interface and interface (yellow) between OpenLB and HYB3D

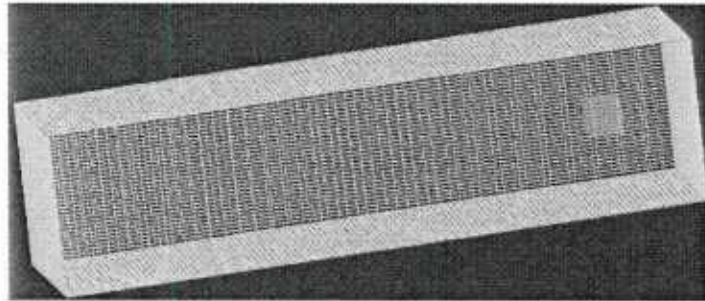


Figure 10. Mesh on surface for HYB3D with interface nodes (gray) for OpenLB

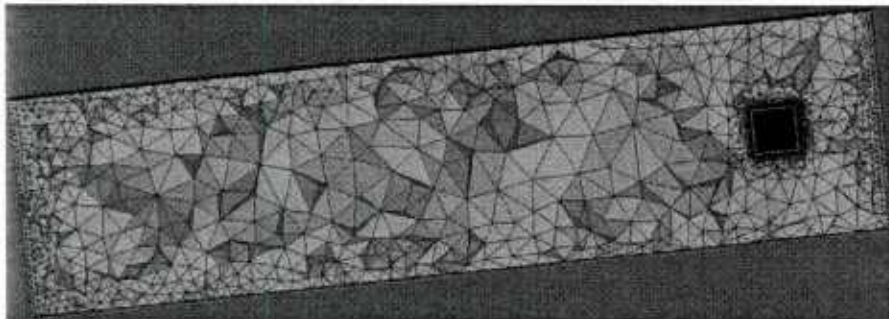


Figure 11. Volume mesh on the cross section in the middle of square duct for HYB3D

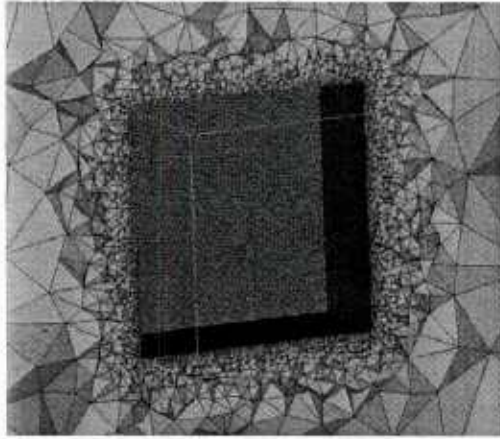


Figure 12. Zoomed interface for OpenLB in the volume mesh for HYB3D

B-2 Physics Based Modeling and Simulation on Graphical Processing Units (GPUs) – Porous Media Flow in Liquid Composite Molding

Authors: R. Haney and R. Mohan, North Carolina A&T State University

Architecture-Performance Interrelationship Analysis in Single/Multiple CPU/GPU Computing Systems: Application to Composite Process Flow Modeling

Abstract

Current developments in computing have shown the advantage of using one or more Graphic Processing Units (GPU) to boost the performance of many computationally intensive applications but there are still limits to these GPU-enhanced systems. The major factors that contribute to the limitations of GPU(s) for High Performance Computing (HPC) can be categorized as hardware and software oriented in nature. Understanding how these factors affect performance is essential to develop efficient and robust applications codes that employ one or more GPU devices as powerful co-processors for HPC computational modeling.

The present work analyzes and understands the intrinsic interrelationship of both hardware and software categories on computational performance for single and multiple GPU-enhanced systems using a computationally intensive application that is representative of a large portion of challenges confronting modern HPC. The representative application uses unstructured finite element computations for transient composite resin infusion process flow modeling as the computational core, characteristics and results of which reflect many other HPC applications via the sparse matrix system used for the solution of linear system of equations. This work describes these various software and hardware factors and how they interact to affect performance of computationally intensive applications enabling more efficient development and porting of High Performance Computing applications that includes current, legacy, and future large scale computational modeling applications in various engineering and scientific disciplines.

CHAPTER 1

Introduction

Recent years have seen the slowing of computational intensity offered by standard **Central Processing Unit** (CPU)-based systems, while scientific/engineering applications have inexorably grown in the need for computational power [1-3]. As the CPU approached maximum power sustainability around 2003, growing from $1\text{watt}/\text{cm}^2$ to $20\text{watts}/\text{cm}^2$ [3, 4], the **Graphics Processing Unit** (GPU) was increasing its computational intensity while maintaining efficient power management [5, 6] and lowering cost to meet the high demand placed upon it from a thriving game industry [1, 2, 7]. The differences in CPU and GPU computational power can be traced back to the designs upon which the two are predicated – i.e., **Instruction-Stream Based** (ISB) and **Data-Stream Based** (DSB) models.

1.1 Background and History

The ISB design of the CPU, whereby a single stream of instructions and data are fed to the device, limited any optimization of arithmetic operations since the input stream could contain any number of potentially complex instructions. Therefore, the CPU accomplished the mitigation of latency by defining elaborate memory caches where processes could be switched out when needed, such as with an I/O interrupt, constraining larger numbers of transistors to the **Memory Management Units** (MMU) and logic device arrays [2, 8, 9] for complex operations such as speculative branching [10-12]. In comparison, the DSB design of the GPU, with instructions and data fed to the device as separate streams, could optimize for arithmetic operations as the instruction stream is committed prior to any data input. Therefore, the GPU accomplished mitigation of latency by pushing as many processes as possible through the device at any given time, conscripting large numbers of transistors for floating-point operations and assumed large arrays of uninterrupted data streams via a wide data bus [2, 8]. The DSB and ISB paradigms define the framework upon which the present computational architectures and software designs for the CPU and GPU have evolved (see Figure 1).

1.1.1 CPU-GPU hardware parallelism.

The CPU computing architecture, as per the ISB model, allowed for the maintenance of increasingly complex processes [9, 13], the execution of which is physically and logically defined by the concept of *pipelines*. CPU *pipelines* are constructs which consist of stages of processing elements executed in a series – each output of a stage is the input to the next [9, 13]. This single pipeline evolved to the more efficient multiple pipelines [9, 13], eventually leading to *super-scalar* systems [14] and *vector processing* machines [15] in an effort to maximize hardware oriented computational power for the CPU-only computing architecture.

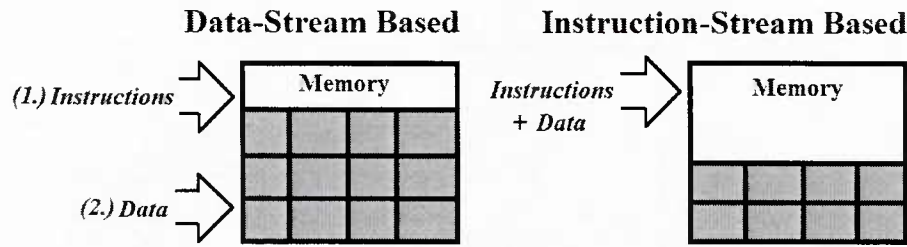


Figure 1. The general models followed by the GPU (left) and CPU (right).

Both the *vector processing* machines [14, 16, 17] and *super-scalar* systems [9, 14] increase the computational power of CPU-only computing architectures using a low-level hardware defined parallelism but take different routes to achieve this goal. The *super-scalar* system defines hardware parallelism via execution of multiple operations per clock cycle augmenting the system with large numbers of registers [9, 14], whereas the *vector processing* machine implements common operations across multiple data elements [14, 16, 17]. Vector processing machines, such as the **CRAY** [18] and the **NEC SX-8** [19], obtain magnitudes of speedup over scalar systems by pushing vector elements onto a special register known as a *pipe* and then execute operations across these elements simultaneously [14, 15]. These CPU architectural designs evolved to coerce a parallelism that is intrinsically and naturally present in the GPU device architecture.

Initially computer graphics were defined as simple vector devices executing as a separate process using **Direct Memory Access** (DMA) to bypass frequent interrupts to the CPU [9, 13], but as the GPU matured into a dedicated device it was free to optimize for throughput, as per the DSB model. The larger concentration of transistors in the floating-point operations coupled with a wide data bus produced the computational intensity for which the GPU computing architecture is widely reputed [20-23] – large numbers of data input is executed simultaneously by preconfigured floating-point operations [2, 8, 24]. The execution of floating-point operations across the set of data input is physically and logically defined by the construct of a *graphics pipeline* (see Figure 2 and Figure 3).

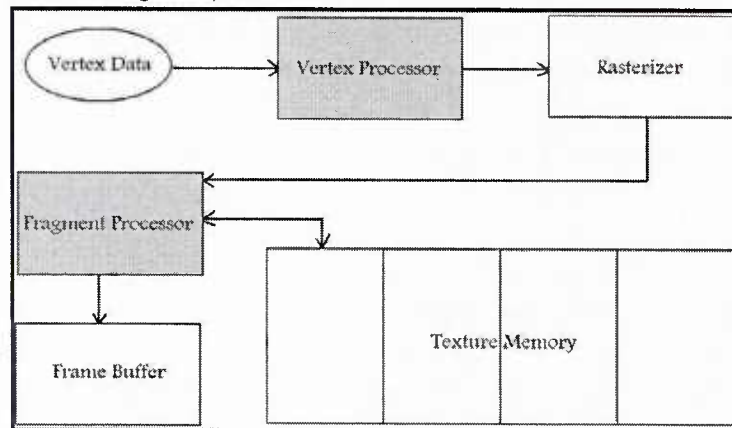


Figure 2. Generic graphics rendering pipeline.

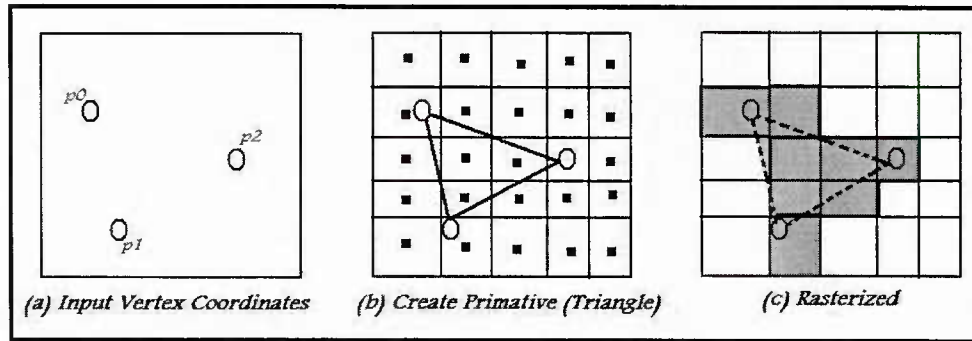


Figure 3. Rasterizing a simple triangle.

The *graphics pipeline* construct is a feed-forward system [24] designed to perform operations on four fundamental entities for final rendering by the display device – vertices, primitives, fragments, and pixels [2]. Initially these fundamental entities were defined wholly within a fixed-pipeline [1, 2] but evolved to become a mix of fixed and programmable sections represented by an interleaved series of *fixed functions* and *programmable stages* across which the data entities flow [1, 2]. Vertex generation is a *fixed function* and is the first step in the graphics pipeline, generating a series of vertices using lists of descriptors from the graphics application [2]. The output vertices from the vertex generation function are passed to the *programmable stage* of vertex processing, generating sets of vertex records independently from each vertex – projecting the original 3-coordinate system to a 2-coordinate system [2]. The input vertices are grouped next into an ordered stream of primitives by the *fixed function*, primitive generation and then passed to the *programmable stage* of primitive processing – potentially merging multiple primitives for rendering [2]. The modified primitives are passed to the fragment generation *fixed function*, producing fragment records that are interpolated from samples of the input and passed to the *programmable stage*, fragment processing [2, 24, 25]. The fragment processing *programmable stage* simulates the interaction of light and surface with the input fragment records – textures are defined at this stage as 1-D, 2-D, or 3-D arrays [2, 24, 25]. The final *fixed function*, pixel operations are next and calculate output pixels for rendering using the input fragment's screen position [1, 2, 7, 24, 25].

The software paradigms have also developed conjointly with the architectures for both the CPU and GPU devices – the two modes evolving as emphasis on higher performance and ease of development have matured.

1.1.2 CPU-GPU software parallelism. The CPU-only software paradigms have evolved along two general modes to increase computational ability – *threaded* and *message passing* [15, 22, 26-28]. The former defines a methodology that concurrently runs independent threads of execution within a single address space [26] and the latter uses message constructs to

communicate and pass data with different processors [26, 28]. These CPU-only software paradigms denote parallelization as a means of boosting application performance – a common practice in HPC applications that has historically been efficacious [26, 28-30].

The actual implementations of the *threaded* and *message passing* systems is defined by several standards, libraries and/or specialized languages that include **Unified Parallel C** (UPC), **OpenMP**, **PThreads**, **Parallel Virtual Machine** (PVM), and **Message Passing Interface** (MPI) among others [26, 28, 29]. Flynn’s Taxonomy categorizes the threaded paradigm into the **Single Instruction Multiple Data** (SIMD) and the message passing paradigm into the **Single Program Multiple Data** (SPMD) as the threaded mode requires a lockstep synchronization in a *shared* memory context and the message passing mode can operate in a *distributed* memory context with varying degrees of autonomy [26, 28].

The GPU software paradigms, unlike the CPU counterparts that explicitly sought to boost performance via parallelism, evolved out of a need to render specialized visuals for graphics heavy applications e.g. high demand games [1]. Once the *graphics pipeline* became flexible, allowing programming of the vector and fragment processors, an efficient set of software constructs for rendering advanced 3-D visuals was needed and *shader languages* and *libraries* were developed, so called because the graphics programs generally were written to *shade* fragments of a given rendered object [24, 31], and included the first portable library - Silicon Graphics ubiquitous OpenGL [1, 7, 32]. OpenGL was a watershed moment in graphics programming as now applications no longer had to be written specifically for a given architecture and/or operating system, rendering and manipulating primitives using sets of matrix operations that included transformation, translation, rotation, and scaling [33, 34].

The OpenGL graphics library was written as an extension of the C language and is built using a basic 4-element vector $\{x, y, z, w\}$ such that if $w = 1$ the vector defines a position in space and if $w = 0$ it is a defined direction [24, 34, 35]. These basic 4-element vectors describe a homogeneous coordinate system with the w element allowing translation and rotation, building into a series of 4×4 matrices that can be modified simultaneously with a single formula [1, 2, 24, 35]. Figure 4 depicts a 16-element operational matrix used by OpenGL such that the 12th, 13th, and 14th elements are the translation components – i.e. M_{03} , M_{13} , and M_{23} in the figure.

$$\begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ M_{30} & M_{31} & M_{32} & M_{33} \end{bmatrix}$$

Figure 4. An example 4-by-4 matrix utilized by OpenGL.

Figure 5 shows a generic 2-D translation of a simple rectangle that occurs after applying translation elements to each of the vertices – done simultaneously by the execution of a translation matrix similar to the model shown in Figure 4.

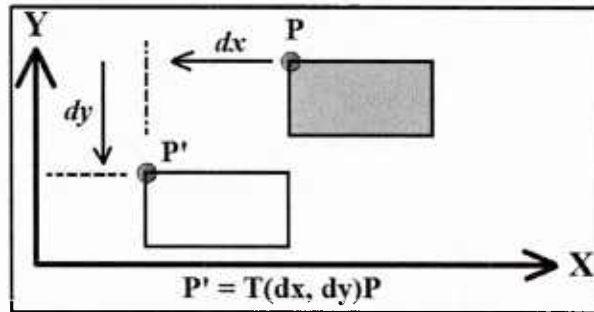


Figure 5. Vertex point P is shifted left and down with translation values.

The OpenGL library defines the matrices as column-major order rather than row-major but this essentially makes no difference as pre-multiplying with row matrices is the same as post-multiplying with column matrices [33, 34].

The advent of OpenGL provided a boost to graphics programming itself, accessing the growing computational power afforded by the GPU for non-rendering purposes was still hindered by the tedious and difficult mappings required by the library. The release of Nvidia's **Compute Unified Device Architecture (CUDA)** in 2007 marked the beginnings of **General Purpose GPU (GPGPU)** computing as it is known today [7, 25, 36]. The CUDA API uses C language bindings to access underlying system calls to the GPU processors and embraces the familiar concept of threading. CUDA utilizes a GCC-like compiler, NVCC, to compile the GPU-bound code to low-level **Parallel Thread eXecution (PTX)** virtual machine and Instruction Set Architecture [37, 38]. PTX provides a machine independent architecture for CUDA compilers to target and allows for portability across multiple GPU generations [37, 38].

Computational modeling and simulations in many fields require both *hardware* and *software* compatibility and influence the resultant computational performance [20, 23, 25, 39-41]. The factors that influence the computational performance can be categorized as *software* and *hardware* oriented, each category can be further demarcated as computational algorithms and data-structures/layouts under software; and architectural designs for single and multiple CPU/GPU under the hardware category.

1.2 Focus and Objective

The focus and objective of this dissertation is to analyze and understand the intrinsic interrelationship between the computing hardware architecture and software variables on the performance of the single and multiple CPU/GPU computing systems shown schematically in Figure 6. These analysis and discussions are based on computationally intensive, unstructured finite element computations for transient composite process flow modeling application. The discussions are organized into single CPU/GPU and multiple CPU/GPU systems. The reminder of the dissertation is presented and organized into chapters as outlined next.

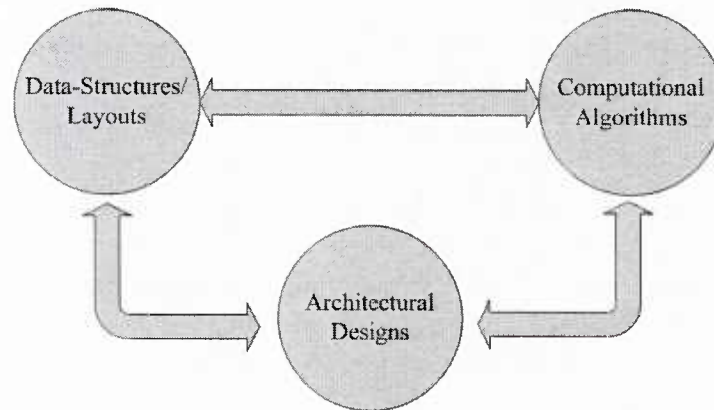


Figure 6. Schematic representation of the interrelationship of computational performance.

Chapter 2 defines the computational problem analyzed and provides key computationally intensive kernels, associated algorithms, and software data-structures/layouts, as well as the hardware descriptions. Chapter 3 establishes the computational potential of the GPU via the analysis and discussion of a key component of many computationally intensive applications – the sparse matrix-vector multiplication. The performance metrics and results as well as the software data-structures and architectural factors that influence the performance are presented. Chapter 4 focuses on the full-solution for the candidate application defining key computationally intensive kernels and associated developments such as data-structures/layouts for the *single* CPU/GPU computing system. The computational performance as it relates to the architecture and software relationship is examined and discussed in this chapter. Chapter 4 will also discuss the hardware architectural factors and problem size and how these work to influence computational performance. All these factors are considered in the development of an empirical computational complexity relationship that will then be correlated to the results and parameters to understand how these factors influence the performance. Chapter 5 is essentially the same as chapter 4 excepting for the use of a *multiple* CPU/GPU computing systems – defining the relevance of these systems for multiple processor designs. Finally, chapter 6 is a summary of the results and analysis as well as proposed future directions for CPU/GPU and hybrid computing systems.

This last chapter will establish the interrelationship between software-hardware variables on computational performance and how these findings can be applied to many computationally intensive engineering and scientific applications, not just the candidate application employed in this work.

CHAPTER 2

Computational Problem – Candidate Application

This chapter describes the computationally intensive physical problem employed as a candidate application for analysis and discussion of computational performance factors – **Resin Transfer Molding** (RTM) process and the transient process resin infusion flow modeling [30, 42, 43]. This candidate application presented employs unstructured finite element computations, assembling sets of locally defined stiffness matrices into a single global stiffness matrix [44], that is sparse, symmetric and positive-definitive [44, 45]. The global stiffness matrix composed of finite element computations, is solved at each time-step during resin infusion flow analysis using the iterative **Preconditioned Conjugate Gradient** method [46] rather than a direct solver such as LU-Decomposition which is computationally prohibitive for most non-trivial problems [44, 45].

The computational model configurations for computational performance analysis built upon finite element meshes employed by the computationally intensive candidate application increase in the problem size based on the 3-noded triangular element and node count but are consistent in geometry and parameters applied – i.e., both mesh models have the same physical descriptions of resin infusion flow modeling excepting size. These meshes, ordered by increasing element/node, are shown below – Figure 7 is representative of both mesh configuration employed in the resin flow infusion modeling analysis for the candidate application.

Unstructured Mesh	Number of Nodes	Number of Elements
MA	26,936	53,178
MB	103,196	204,970

The finite element models defined by meshes **MA** and **MB** are consistent in geometry and physical problem parameter input therefore for exhaustive purposes another, less regular, mesh is analyzed for performance behavior and is denoted as **10FT** – shown below is the number of nodes/elements for this model mesh configuration. The **10FT** unstructured mesh falls roughly into the category of medium-sized computational problem with respect to the other two meshes studied in this research and is composed of 3-noded triangular elements; however, it is more complex in structure. The **10FT** unstructured mesh model is shown in Figure 8.

Unstructured Mesh	Number of Nodes	Number of Elements
10FT	29,171	58,187

Figure 9, Figure 10, and Figure 11 show plots of the non-zero element sparsity pattern of matrices that result from the input meshes **MA**, **MB**, and **10FT** respectively. Clearly meshes **MA** and **MB** are more regular than the matrix from mesh **10FT**, which is though diagonally dominant and symmetric, has non-zero elements dispersed more evenly across the entire matrix.

All algorithms referenced in this chapter are presented in **Appendix C** of this dissertation for convenience and better formatting of text.

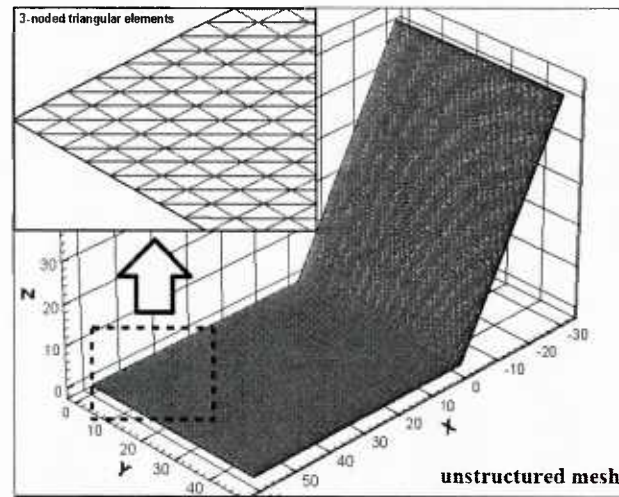


Figure 7. Unstructured mesh geometric configuration used by candidate application.

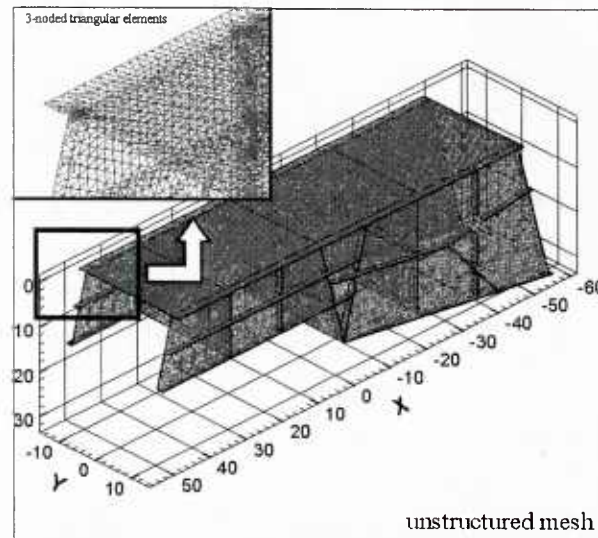
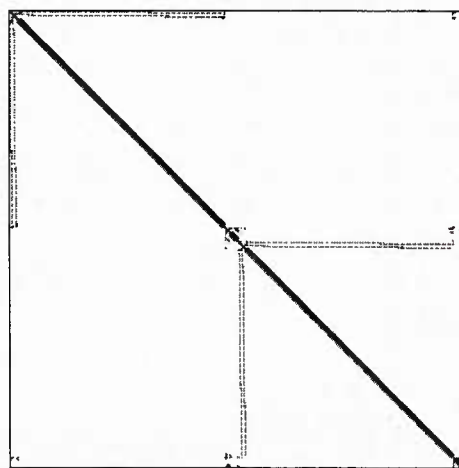
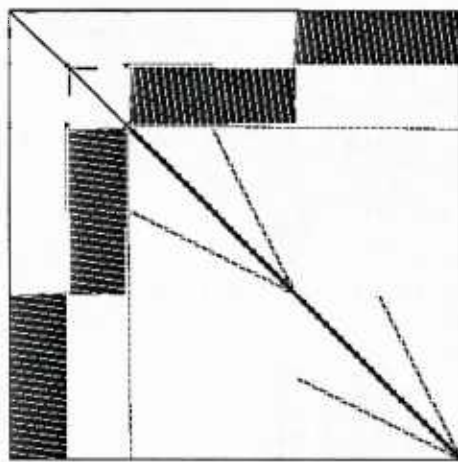


Figure 8. Unstructured mesh geometric configuration of **10FT** model.



Nonzeros = 474912 (0.065%)

Figure 9. Sparse matrix non-zero entry distribution for mesh **MA**.



Nonzeros = 1773270 (0.017%)

Figure 10. Sparse matrix non-zero entry distribution for mesh **MB**.

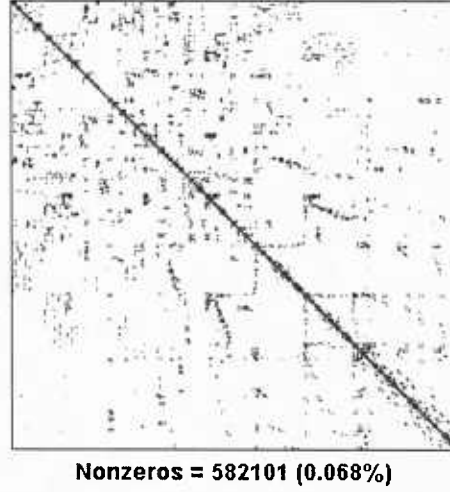


Figure 11. Matrix sparsity pattern of mesh 10FT.

2.1 Physical Description

The **Resin Transfer Molding** (RTM) process flow modeling methodology studied is based on the work published in [30, 43, 47] and is presented briefly next.

2.1.1 Resin Mass Conservation. Following the discussions in [30, 43, 47], the resin flow through the fiber preform contained within the mold cavity is represented by the transient mass conservation equation. The physical mass conservation equation (formed by coupling the mass conservation equation with the momentum equation via Darcian velocity field) is given by equation (2.1) with \bar{K} the permeability tensor, μ the resin viscosity, P the pressure field, and Ψ the state variable representing the infused state of the resin – further details are available in [30, 43, 47].

$$\frac{d}{dt} \int_{\Omega} \Psi d\Omega = \int_{\Omega} \nabla \left(\frac{\bar{K}}{\mu} \nabla P \right) d\Omega \quad (2.1)$$

The value of the state variable Ψ is 0, where the resin has not infused the fiber preform, and 1, where the resin has completely infused the fiber preform in any given region of the Eulerian mold continuity domain Ω used in the **Finite Element Method** (FEM) computations.

As discussed in [43] the application of Galerkin weighted residual formulation and approximating for the pressure P and fill factor Ψ , with appropriate elements and associated shape functions, yields a semi-discrete system of equations given by equation (2.2) with C the

lumped mass matrix representing pore volume, K the stiffness matrix, q the load vector, and $\dot{\Psi}$ the time derivative term.

$$C\dot{\Psi} + K P = q \quad (2.2)$$

The transient semi-discrete equation is then solved by introducing the finite difference approximation given by equation (2.3).

$$\dot{\Psi} = \frac{\Psi^{n+1} - \Psi^n}{\Delta t} \quad (2.3)$$

2.1.2 LCM Solution Strategy The semi-discrete equation (2.2) can be reduced to the equation (2.4) as discussed in [43], with C taken to be the lumped mass matrix.

$$C_{ii}\Psi_i^{n+1} - C_{ii}\Psi_i^n + \Delta t K_{ij}P_j = \Delta tq_i \quad (2.4)$$

The above form of the discretized equation is solved by the LCM process flow simulation algorithm at each time-step. Equation (2.4) defines the implicit form of the process flow modeling in LCM detailed in [43]. The generalized algorithm for the finite element approximation of the process flow modeling for each time step is summarized in **Algorithm 2.1** - the interested reader is referred to [30, 43, 47] for further details on the LCM process and its conversion to the Finite Element Method formulation for the computational simulation.

2.2 Key Computationally Intensive Functions

Examination of **Algorithm 2.1** reveals the most computationally expensive function of the candidate application – the solution to the system of linear equations given in matrix form as $\underline{Ax} = \underline{b}$ embodied on line 9 as $\left[\hat{K}_{ij} \right]_m \{P_j\}_m = \{g_i\}_m$. The solution to systems of linear equations can be accomplished via direct methods or iterative methods - each has advantages and disadvantages.

Direct solution methods such as LU-Decomposition provide a solution in a single direct solution step for the linear equation system without a need of initial guess solution vector but require high computational costs whereas iterative methods start with an initial guess that iteratively converges to a solution vector but have significantly lower costs [45]. The candidate application presented executes this solver within a nested loop so any incurred costs from a single call to the solver will be exacerbated by the product of $(K \times L)$ with K the number of iterations for mass-convergence and L the number iterations for all nodes to be determined as *filled* – in addition to the iterations for the convergence of the iterative solution. The **Preconditioned Conjugate Gradient (PCG)** iterative solver was selected as a balance of computational cost and accuracy for the solution of linear system of equations.

2.2.1 The Iterative Solver. The PCG iterative solver chosen for the candidate application provides good balance of computational cost and solution accuracy, composed of a set of matrix-

vector operations [44, 46]. The majority of the computational cost of the PCG iterative solver is well documented as the **Sparse Matrix-Vector** (SpMV) multiplication [17, 46, 48-51] – this operation is shown on lines 6 and 9 of **Algorithm 2.2**. The SpMV operation has high computational cost due to three performance issues – poor *locality*, poor *instruction mix*, and a high *memory overhead* [45, 52]. Poor *locality* of SpMV results from indirect and often irregular memory accesses, poor *instruction mix* is derived through the execution of three memory loads for every two floating-point operations, and the high *memory overhead* is due to the largest portion of the matrix being zero and thus useless to the computation but held in memory regardless. The high memory overhead of this candidate application, and indeed any application that builds large sets of sparse matrices, is defined as a *memory-bound problem* and requires data compression data-structures/layouts to execute [30, 44, 48].

There are different compression formats that are used to execute *memory-bound* problems and a discussion of these follows.

2.3 Data-Structures/Layouts

One of the most common form of data compression for sparse matrix data structures used in engineering/scientific application codes for **High Performance Computing** (HPC) is the **Compressed Sparse Row** (CSR) structure which is partly due to its ease of programming [52-54]; however this is not the only model to follow and much research has been done to explore this area as the format used can have significant impact on the resulting performance [44, 49, 52]. The CSR data compression format consists of three arrays – non-zero data elements, column indices, and row pointers, and operates by iterating over all the rows of the compressed sparse matrix (see Figure 12). Each row is represented as the length between each non-zero element held at the row pointer index currently being iterated over, which is then passed as an input to the column indices array resulting in the original (*row, column*) position of the element from the original sparse matrix – now held as an array of non-zero elements [52].

Algorithm 2.3 is the expression of the sparse matrix-vector multiplication using the CSR data compression format [48, 52]. The utilization of the CSR compression format lowers the memory overhead for the sparse matrix-vector multiplication but does nothing for the other two components that effect performance, e.g. *locality* and *instruction ratio* - line 9 of **Algorithm 2.3** embodies an operation that is deleterious to any locality and does not improve the instruction ratio.

The **Block Compressed Sparse Row with 2x2** (BCSR2x2) sub-blocks has been shown to improve the often disappointing performance of the CSR by increasing locality via reduced number of memory loads [21, 52, 55], however this improvement is not guaranteed. BCSR2x2 operates on sets of sub-blocks of dimension 2x2 rather than single elements, but locality is only

improved if the original sparse matrix is composed of dense sets of sub-blocks; otherwise memory overhead is increased with no corresponding performance boost (see Figure 13). Determination of these dense sub-blocks of the sparse matrix is not known until runtime, making this a dynamic problem.

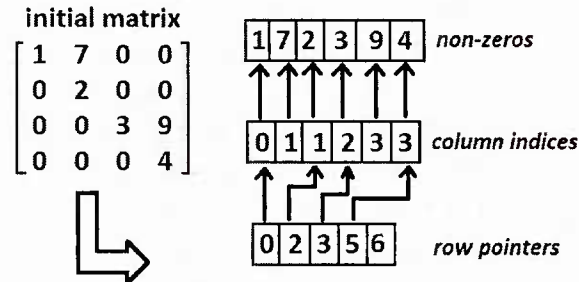


Figure 12. The CSR format.

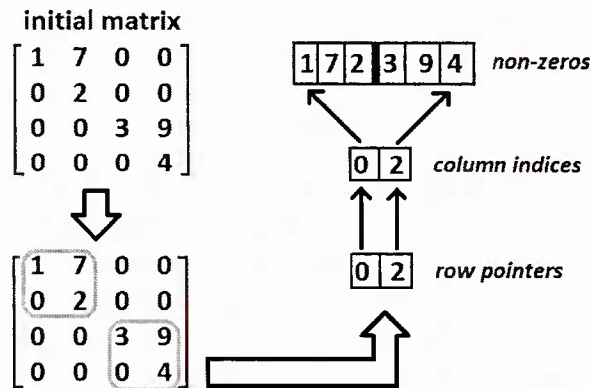


Figure 13. The BCSR2x2 format.

2.4 Hardware Discussions of Computing Architectures Used

The GPU, CPU code developments from the present work for the candidate application is executed within the context of two separate computing machine architectures generally categorized as CPU-based, GPU-based, and/or some combination of the two. **Biomedical Analysis and Simulation Supercomputer** (BASS) located at Chapel Hill, North Carolina is categorized as **System A** for the remainder of this dissertation, and system called as **OAKLEY** located at Ohio Supercomputing Center in Columbus, Ohio is categorized as **System B** for the remainder of this dissertation. The **System A** CPU/GPU computing system is composed of AMD Dual-Core Opteron CPUs and Nvidia Quadro FX 5600 GPUs each with a 2.8 GHz and 1.5 GHz clock frequency respectively. The **System B** CPU/GPU computing system is composed of Intel Xeon X5650 6-Core CPUs and Nvidia Tesla M2070 GPUs each with a 2.66 GHz and 1.15 GHz clock frequency respectively. The details of the hardware designs for both computing systems are discussed next – categorized separately by CPU and GPU.

2.4.1 CPU Hardware Architectures The CPU hardware designs for **System A** and **System B** follow the same concepts, but **System B** is more advanced (see *Table 1* and *Table 2*). These architectures both have multiple levels of cache memory, 2 levels for **System A** and 3 for **System B**, and different capacities [56, 57]. **System B** has greater cache memory capacity at all levels, and while both computing systems have the same memory I/O width, **System B** has faster memory devices – double that of the memory devices for **System A** [56, 57].

Both **System A** and **System B** have embraced the concept of multiple core architectures, but as with the memory structures **System B** is the more advanced of the two. A processing core can be viewed as a separate CPU executing in the same address space enabling the computing architecture to theoretically increase computational ability [9, 13], each core operates at a global clock frequency for the device – **System B** executes 6 processing cores whereas **System A** has 2 [56, 57]. **System A** executes these cores at a higher clock frequency but **System B** has three times the number of cores – a faster design for the hardware level.

Table 2
System A – CPU hardware architecture

Processor Clock	2.8 GHz
Processing Cores	2
L1 Cache	256 KBs
L2 Cache	2 MBs
Memory I/O	64-bit DDR2 SDRAM

Table 3
System B – CPU hardware architecture

Processor Clock	2.66 GHz
Processing Cores	6
L1 Cache	384 KBs
L2 Cache	1.50 MBs
L3 Cache	12 MBs
Memory I/O	64-bit DDR3 SDRAM

2.4.2 GPU Hardware Architectures. The GPU hardware designs for **System A** and **System B** follow the same concepts, but **System B** is more advanced (see *Table 3* and *Table 4*). These architectures both have processing cores in the hundreds to accommodate the massive computational power involved in optimizing throughput, however **System B** has more than three times the number offered by **System A** [58, 59]. **System A** has a faster clock frequency for each processing core but has a significantly lower number of these cores so the aggregate power of **System B** is greater regardless [58-61]. Each of these cores can be viewed as a separate

processor, but unlike the CPU, the GPU will execute the same instruction for all input until a reconfiguration occurs for the next large set of input [31, 35] hence less emphasis on memory management via cache hierarchies.

Both **System A** and **System B** have wide memory I/O at 384-bits to sustain the high number of processes required to maximize throughput as a latency mitigation strategy prescribed by the DSB paradigm which the GPU follows. The memory device employed by **System B** uses the same input width but is faster than that of **System A** [58, 59, 61].

System B is part of the CUDA Compute Architecture 2.0, a metric used by Nvidia to categorize the device versions at both the software and hardware levels, whereas **System A** is 1.0 – a significant difference [37, 62]. The higher Compute Architecture of **System B** allows more options, e.g. Nvidia set of libraries, and **System B** was the *first* GPU device to fully embrace general programming on the GPU (GPGPU) as it contained no video outlet, had double-precision abilities, and provided Error-Correcting Code (ECC) [25, 37, 62]. The GPU device architecture for **System B** is superior to that of **System A**.

Table 4
System A – GPU hardware architecture

Processor Clock	1.35 GHz
Processing Cores	128
Memory I/O	384-bit GDDR3
Register Count	8,192
Shared Memory Banks	16
CUDA Compute Architecture	1.0

Table 5
System B – GPU hardware architecture

Processor Clock	1.15 GHz
Processing Cores	448
Memory I/O	384-bit GDDR5
Register Count	32,768
Shared Memory Banks	32
CUDA Compute Architecture	2.0

2.4.3 Hardware Design Summary The hardware designs for both systems presented reflects the general concept of the DSB and ISB paradigm for the CPU and GPU respectively resulting in different latency mitigation policies. Both **System A** and **System B** utilize cache memory hierarchies to mitigate latency for the CPU designs and both have processing core

counts in the hundreds to optimize throughput for the GPU designs, as per DSB paradigm. The memory I/O bus width is higher for the GPU hardware than for the CPU hardware, while they are equal in size to one another – **System B** has a faster memory device in both cases. **System B** is a more advanced hardware design for both the CPU and GPU devices in context of processing and memory. The next chapter will discuss the computational potential offered by the GPU via performance of the highest cost operation of the candidate application, as well as many HPC scientific and engineering computational modeling applications [20, 30, 48, 49, 63], the sparse matrix-vector multiplication.

CHAPTER 3

Computational Potential of GPU – Sparse Matrix-Vector Multiplication

This chapter focuses on the computational *potential* offered by the GPU for computationally intensive applications such as the candidate application presented. The candidate application has highest computational cost at the point of the solution to system of linear equations that are presented in matrix form as $\underline{Ax} = \underline{b}$. This sparse matrix system is solved iteratively using PCG in deference to the computationally prohibitive costs of using direct solver methodology – a common practice in HPC modeling applications [20, 30, 48, 49, 63].

The PCG solver is composed of matrix operations that should map well to the GPU device given it was created to execute massive numbers of matrix operations in tandem [1, 2, 7, 24]. The Sparse Matrix Vector Multiplication (SpMV) operation embodies the highest computational cost of the PCG solver, up to 90% of the total cost [25, 29, 48, 49, 63] - the minimization of this operational cost will provide a significant performance boost to the presented candidate application as well as many other HPC applications. However, mapping the SpMV operation to the GPU involves software factors, including software API CUDA, that are intimately related to underlying hardware architectures provided during the computational modeling application execution.

3.1 Mapping Sparse Matrix-Vector Multiplication to GPU

The SpMV operation is the highest cost component of the presented candidate application, as well as many HPC applications that employ systems of sparse matrices and the potential boost provided by utilizing the GPU as a co-processor has generated a lot of interest [21, 48, 49, 53, 54, 64]. The SpMV derives its high operational cost not from floating-point operations, as its *instruction mix* is poor [48]; rather the inefficient memory accesses are

culpable. Properly mapping such a high cost and computationally weak operation to a computationally powerful device like the GPU for optimal performance involves an understanding of the software as well as the underlying hardware. The CUDA API is the software API employed by the presented candidate application and is representative of this relationship. This is discussed in the context of the actual mapping of the SpMV algorithm to the CPU/GPU computing system next.

3.1.1 The CUDA Software/Architecture API. The Nvidia CUDA Software API for general GPU programming is both a software and hardware system that uses higher level C language extensions to call lower-level OpenGL/DirectX libraries – accessing the GPU device [25, 37, 62]. CUDA maintains the C language concept of *threads* [62], but unlike the C language CUDA exposes the *memory hierarchy* to the developer [25, 37] which is a necessity as the GPU has no real virtual memory system and remains as flat as possible for the optimization of throughput. Despite the added complexity of an exposed memory hierarchy, CUDA freed developers from the necessity of translating general code to and from data-structures that the GPU understood i.e. column-major matrix operations, complete with model-view and matrix-stacks so necessary for lower-level graphics programming [31, 33-35].

CUDA GPU threads, memory, basic API syntax, and associated device hardware architecture are discussed in the following sub-sections.

3.1.1.1 CUDA API thread hierarchy The GPU thread as defined by the CUDA architecture is different from the **Light Weight Process** (LWP) familiar to Operating System design [9, 37, 62]. The GPU has zero-cost context switching of threads because this is executed at the hardware level, commonly known as *hardware multi-threading*, whereas the Operating System paradigm of threads is one controlled by software either in user or kernel space naturally incurring overhead. Figure 14 from Nvidia [62] illustrates the hierarchical structure of threads.

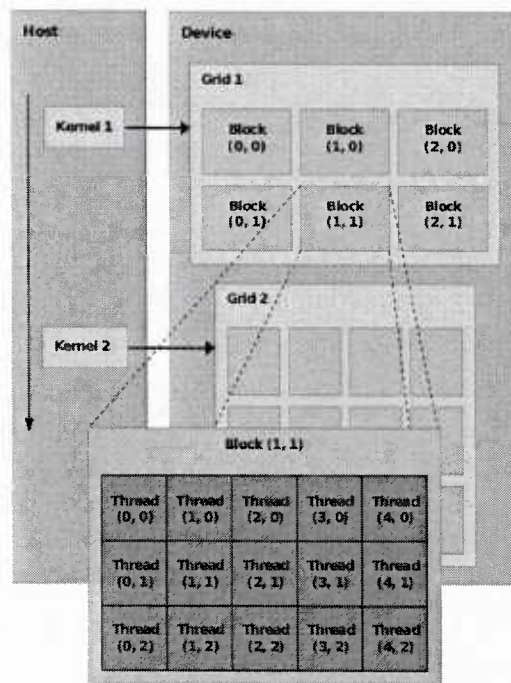


Figure 14. Diagram of CUDA GPU thread grouping.

The CUDA paradigm provides a direct mapping from logical constructs, such as the thread, to hardware architecture designs as shown in Figure 15. The large number of transistors dedicated to floating-point operations is grouped together as sets of **Streaming Processors** (SPs), defining the thread construct [62]. These SPs are grouped in sets of 8 to form sets of **Streaming Multiprocessors** (SMPs) – the domain of the Block [62]. The largest logical construct, the Grid, is embodied by the groupings of SMPs [62] as shown in Figure 16.

Hardware	Logical Construct
Streaming Processor	Thread
Streaming Multiprocessor	Block
Chip/Device	Grid

Figure 15. CUDA hardware to logical construct mapping.

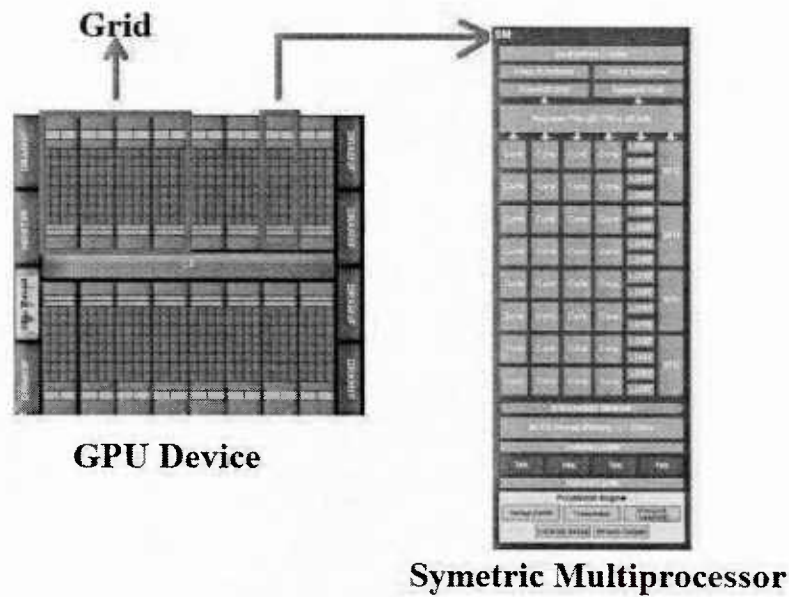


Figure 16. CUDA hardware with logical Grid overlay.

CUDA executes operations via 24-stage graphics pipelines each fully completing in 4 memory clock cycles using a single SMP defining the logical unit of execution as batches of 32 threads called a *warp* - $4 \times 8 = 32$ [37]. The CUDA paradigm increases parallel granularity, naturally extending from the single thread to the Block which is composed of sets of threads, to the Grid which is composed of sets of Blocks [37].

All threads in a Block are assigned to a single SMP, abstracted from the programmer, although multiple Blocks can be assigned to a single SMP [65]. The abstraction of the Block/thread/SMP construct is the dominate strategy to produce scalability of CUDA to different generations of GPU devices – the programmer need not know the exact number of SMPs to develop GPU-bound code as the hardware will schedule as needed [25, 65].

3.1.1.2 CUDA API memory hierarchy CUDA defines varying layers of memory that reflect both the underlying hardware architecture and logical threads [62]. Figure 17 from Nvidia, shows the overview of logical memory constructs to the underlying physical hardware – clearly the GPU is not equivalent to the CPU in memory complexity but does provide some level of layering [1, 2, 62].

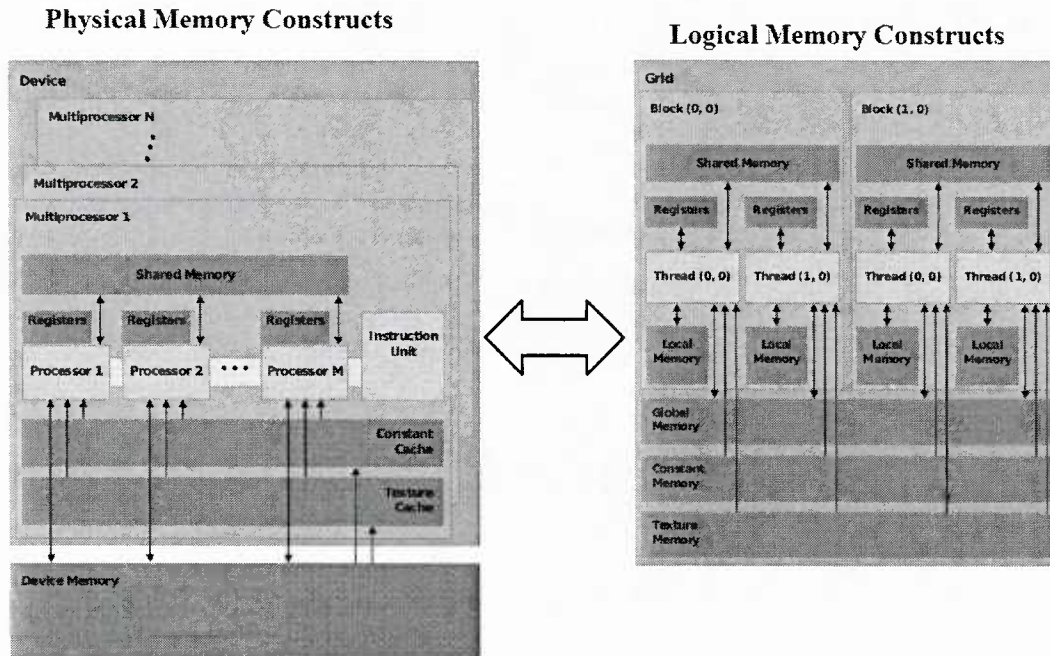


Figure 17. CUDA GPU mapping of the physical and logical memory structure.

The lowest level of memory in the defined hierarchy is the register file, composed of the set of registers for the SMP – each thread has mutually exclusive access to an on-chip register and local memory in read/write mode [2]. As with CPU hardware designs, the register is the fastest of the two on-chip structures with local memory costing approximately 20 to 50 clock cycles [1, 2, 62, 66]. The next highest level of memory for the CUDA design is the shared memory structure. Shared memory is bound to a given Block and each thread has read/write access implying the need for synchronization to avoid race conditions [1, 2, 9, 62, 66] – the next set of memory levels are visible to all Blocks defined in the application.

Constant and texture memories are both read-only with regards to the threads in any given Block but texture can yield some level of locality as the CPU has write access to this structure [2, 24]. Global memory, sometimes called device memory, has the highest capacity and clock cycle cost running as high as 600 to 800 cycles per call [1, 2, 62, 66] – global memory is the only area where the CPU and GPU can communicate using the Peripheral Component Interconnect Express (PCIe) bus. The PCIe bus is a well-known point of bottleneck in many CPU/GPU computing HPC applications employing sparse matrix systems [48, 53, 64, 67, 68].

3.1.1.3 CUDA API basic syntax The CUDA API is an extension of the C language invoking its own GCC –like compiler, e.g. NVCC, to compile high-level GPU Kernels to PTX machine independent code which is executed at runtime [38, 62] – CUDA recognizes GPU-bound code via keywords. These keywords are prepended to the C-like function signatures [25,

62] defined as the GPU-bound Kernel and prior to calling must have memory set aside for the execution of each on the GPU device – as a `<<< Blocks, Threads >>>` structure. Figure 18 illustrates an example usage of the GPU Kernel declaration and memory space allocation for a generic function.

Function signature for CUDA Kernel with global keyword

```
__global__ void myKernel (void)
```

•
•
•

Calling Kernel with 400 Blocks composed of 128 Threads each

```
myKernel<<< 400, 128 >>> ( )
```

Figure 18. Example defining CUDA Kernel function signature and execution space.

3.1.2 Algorithmic Strategies for SpMV Mapping. Mapping the SpMV operation to the GPU via the CUDA Software API presents the immediate challenge of how to distribute the matrix to the set of *warps* to be executed. A straightforward approach would apply a single thread per row, chunking the domain into sets of 32 – this is not the best approach as documented in [69]. As with [69], the SpMV operation is mapped to the GPU device using the one warp per row concept to obtain better utilization of the device resources – this is discussed in detail after the initial performance results.

Details of the code used in this chapter to gather the performance results are presented in **Appendix A** and includes both the CSR and BCRS2x2 data compression formats.

3.2 GPU Initial SpMV Performance Results

The results of SpMV on the CPU/GPU for both machine system architectures, **System A** and **System B**, using the CSR data format are presented in this section with the goal of exposing the performance effects of software, hardware, and algorithmic factors using a consistent model in differing computing environments. These results are gathered using randomly filled sparse matrices with 50% sparseness, increasing in total matrix sizes from 1K to 4K.

Critical to understanding the observed results is the establishment of *metrics* to define performance benchmarking. Computational performance benchmarking for the GPU and CPU developments and resin flow infusion modeling for the remainder of this dissertation was accomplished as follows.

- **Normalized FLOPS:** The raw count of floating-point operations is modified by clock frequency of the device being measured to address the variance in processor speeds for the GPU and CPU architectures. Equation (3.1) illustrates the normalization process described, denoted as $FLOPS_{norm}$, with $FLOP_{cnt}$ the raw count of floating-point operations, C the clock frequency, and T the total execution time.

$$FLOPS_{norm} \equiv \frac{FLOP_{cnt} \times C}{T} \quad (3.1)$$

- **KFLOPS:** The approximate thousands of floating-point operations per second. As the GPU and CPU architecture vary in their processor speed, the KFLOPS is normalized by the clock cycle of the device being measured.
- **Speedup factor:** The ratio of CPU execution time to GPU execution time whereby the larger the value, the more optimal the performance obtained through the GPU.

System A is the first architecture examined followed by **System B**.

3.2.1 System A. The CPU/GPU computing system execution of the SpMV operation was compared against the CPU-only version. The CPU-only environment is much slower than that of the CPU/GPU environment in every case as shown in *Table 5*. The CPU/GPU computing system increases in performance at an almost exponential rate, accelerating at the 2K matrix – this is consistent with previous findings of GPU performance on larger input models [1, 21, 22, 70].

Table 6

Time comparison for SpMV on System A (CSR compression)

Matrix Rows	CPU Time (ms)	GPU Time (ms)	Speedup Factor
1024	9.479572	0.167552	56.5769
2048	35.138212	0.069408	506.2559
4096	148.453443	0.06912	2147.764

3.2.2 System B. The CPU/GPU computing system execution of the SpMV operation was compared against the CPU-only version. The CPU-only environment is much slower than that of the CPU/GPU environment in every case as shown in *Table 6*. The CPU/GPU computing system increases in performance at an almost *exponential* rate, accelerating at the 2K matrix – this is consistent with previous findings of GPU performance on larger input models [1, 21, 22, 70].

Table 7

Time comparison for SpMV on System B (CSR compression)

Matrix Rows	CPU Time(ms)	GPU Time(ms)	Speedup Factor
1024	1.6948	0.059648	28.41336
2048	8.417524	0.055424	151.8751
4096	30.3342	0.047872	633.6522

3.3 Software Data-Structures/Layouts Factors

The previous section of this chapter establishes an obvious benefit in performance when using the CPU/GPU computing system over CPU-only in both **System A** and **System B** computing environments. However it is necessary to understand how different software data-structures/layouts can affect performance in CPU/GPU systems in order to optimize for computationally intensive applications. The first software factor to be analyzed is one that is commonly touted in the GPGPU community – *thread occupancy* [37, 49, 71, 72].

The thread occupancy of a CUDA enabled GPU device is defined as a ratio of active *warps* to the maximum number of *warps* supported by the Compute Architecture [37] – **System A** which is Compute Architecture 1.0, and **System B** which is Compute Architecture 2.0 support 24 and 48 *warps* per SMP respectively [58-61]. The importance of thread occupancy can mean the difference of as much as *20-times* performance boost [37, 73] . However, arbitrarily assigning the largest number of *warps* per block possible is the wrong approach.

There exists a finite set of registers that are allocated for each of the thread blocks, and if each block requires many registers as defined by threads, the aggregate number of active blocks possible is reduced and correspondingly the occupancy is reduced and performance suffers [25, 37]. For example, **System A** defines 8,192 32-bit registers for each SMP and can execute at most 768 threads meaning that at most $\frac{8192}{768} = \lfloor 10.6666... \rfloor \approx 10$ registers can be used per thread to achieve 100% occupancy. The negative effects on performance can be further compounded by register *spilling* to device memory, increasing memory cycle counts hundreds of times [37, 62]. Both CPU/GPU computing systems architectures were determined to obtain maximum thread occupancy at 256 threads per block, a multiple of the *warp* size – providing the optimal access to local registers and avoiding costly code spills, allowing the hardware to properly coalesce memory addresses [62, 69].

Another software factor that can affect performance of a CPU/GPU computing system is the data compression format used – understanding this is vital to optimizing memory-bound applications such as the presented candidate application [2, 52, 74, 75]. As noted in the

introduction, the SpMV lends itself to several performance challenges – key to the data compression format is locality. The CSR data compression format has poor locality due to frequent address indirections and BCSR2x2 can mitigate this by lowering the number of memory loads per floating-point operation – simply by maintaining a 2x2 sub-block set rather than a single element [2, 52, 74, 75]. However, the benefits of using BCSR2x2 rely heavily on the existence of dense 2x2 sub-blocks in the original sparse matrix.

The software factors of thread occupancy and data compression format were combined and executed on a randomly generated 4K sparse matrix defined with a 50% sparseness for both **System A** and **System B**. *Table 7* and *Table 8* shows the performance of these software factors for **System A** and **System B** respectively - Figure 19 and Figure 20 illustrate graphically the same results. Both **System A** and **System B** display increased performance as the number of threads per block grows evidence of better utilization of GPU computational resources. However, dramatic performance increases generated by growing thread occupancy occur at 32 for **System A** and 64 for **System B** shown by Figure 19 and Figure 20 respectively – due to clock cycle execution which is explained in greater depth in section 3.5 of this chapter. The effect of changing data compression format from CSR to BCSR2x2 is greater for **System A** (see Figure 19) than for **System B** (see Figure 20) as the later defines an on-chip cache relegating locality mitigation to a lower impact factor for performance. These results clearly illustrate that software factors can have an impact on the CPU/GPU computing system's performance – the associated hardware factors are discussed next.

Table 8

SpMV time comparison per thread occupancy and data format (System A)

Threads Per Block	GPU Time (ms) - CSR	GPU Time (ms) - BCSR2x2
16	43.0042	15.7161
32	0.1456	0.122944
64	0.068736	0.125344
128	0.080096	0.1272
256	0.06912	0.179392

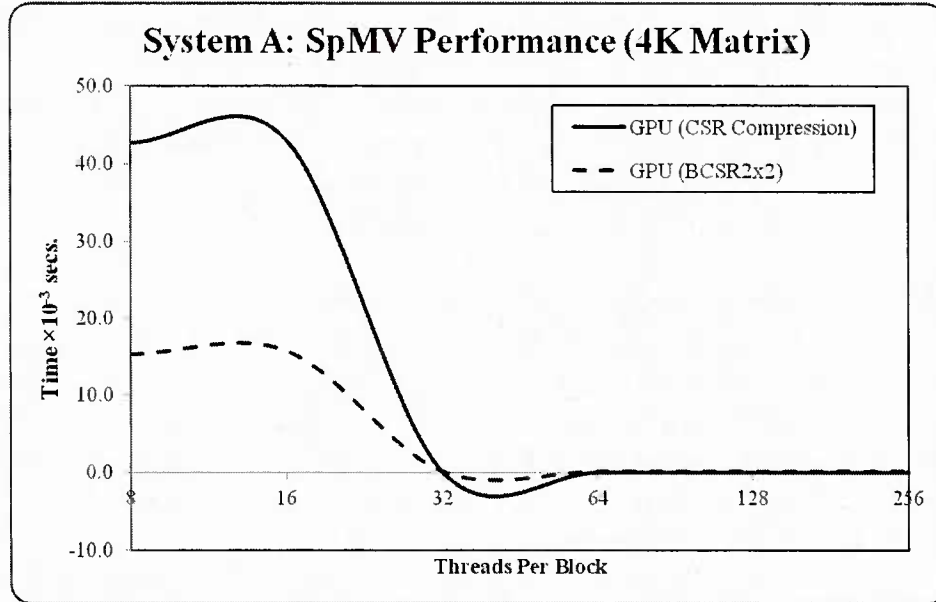


Figure 19. System A performance as threads per block increase (CSR and BCSR2x2).

Table 9

SpMV time comparison per thread occupancy and data format (System B)

Threads Per Block	GPU Time (ms) - CSR	GPU Time (ms) - BCSR2x2
16	18.6252	15.6434
32	32.1774	29.8709
64	0.057152	0.06192
128	0.06928	0.063936
256	0.047872	0.062208

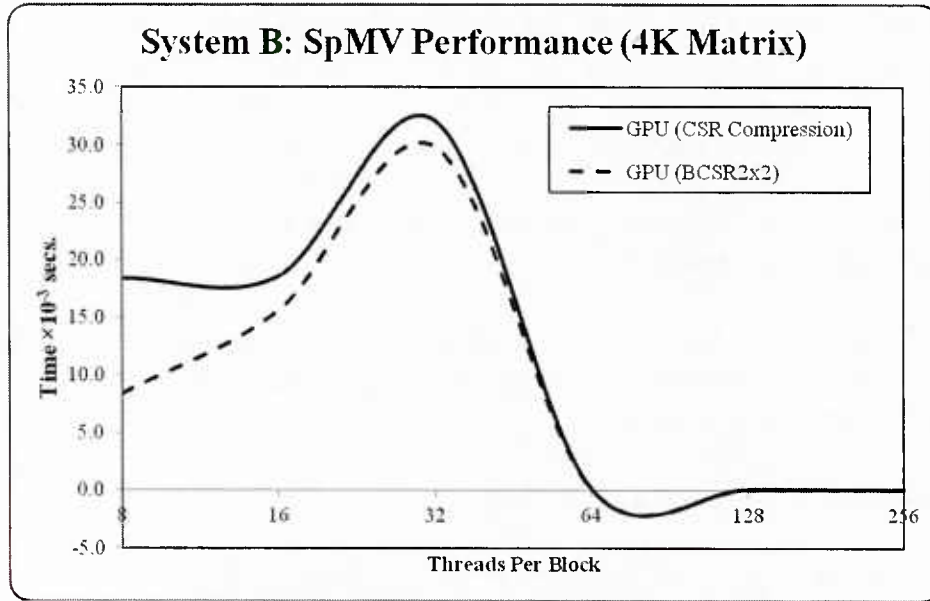


Figure 20. **System B** performance as threads per block increase (CSR and BCSR2x2).

3.4 Architectural Hardware Factors

The CPU/GPU computing systems execute within differing environmental context for **System A** and **System B** and are important components in the resulting performance of SpMV. **System B** has a more advanced CPU and GPU architecture than **System A**, a 6-core CPU and *Fermi* GPU design versus a 2-Core CPU and *Quadro* GPU design for **System B** and **System A** respectively [56-61, 76]. In and of itself, this difference is irrelevant however when comparing **System A** to **System B** as these architectural hardware variations must be factored into the result.

The architectural hardware design of **System B** defines a GPU device that provides extra hardware for context switching compared to the corresponding GPU device of **System A**. The increased switching hardware of the GPU device of **System B** is the manifestation of the *double-pumped* graphics pipeline described by the *Fermi* architecture [61]. Important to the sheer computational abilities are the number of processing cores of **System B** with 448 as compared to **System A** at 128 [61, 76] – the GPU device on **System B** has greater than 3-times the power of **System A** by this metric.

The architectural hardware design of **System B** defines a more advance memory structure than the corresponding structure of **System A** and this is reflected consistently at every memory device [58, 60, 61]. **System B** and **System A** both have 384-bit wide memory I/O but **System B** has the faster GDDR5 memory versus **System A** with GDDR3 memory. The GDDR5 memory of the GPU device of **System B** operates at twice that of GDDR3 – therefore throughput of data

will be maximal for **System B** [58, 59]. The hardware design of the GPU device for **System B** has 4-times the number of registers than the corresponding GPU device on **System A** at 32,768 to 8,192 for **System B** and **System A** respectively [58, 59] – these extra registers will provide more capacity for threads of a given *warp* as a register is thread-bound in nature [25, 66, 69]. Finally, the shared memory of the GPU device of **System B** is 3-times greater than that of **System A** at 49,152 to 16,384 bytes for **System B** and **System A** respectively – providing larger cache-like structure for **System B** [61, 76].

These hardware, software, and algorithmic factors when analyzed individually are important but it is within the context of the aggregate that the real importance is revealed. This interdependence of factors is discussed next.

3.5 Interdependence of Software and Hardware Factors

The previous sections of this chapter have established the importance of software, hardware, and algorithmic factors on resulting performance; however these factors are defined as interdependent. These factors work both independently and with one another to produce the observed performance results for SpMV in this chapter.

Executing one *warp per row* rather than one *thread per row*, the dominant algorithmic factor with regards to mapping the SpMV operation to the CPU/GPU computing system provides a fuller utilization of the GPU device [37, 48]. The fuller utilization of the GPU is directly impacted by the hardware as the entire *warp* is now performing useful work and memory addresses are likely coalesced [37, 48]. The algorithmic factor is also impacted by the software factor of increasing thread occupancy during SpMV as memory is set aside in units likely to increase the use of more threads per warp.

The software factor described by thread occupancy and data compression formats effect performance by increasing memory address coalescing and increasing locality – essentially altering the number of memory loads to the corresponding floating-point operations for SpMV. However, software factors are tied to the hardware with thread occupancy in the same way that the one warp per row is affected, and the impact of the data compression format change was less pronounced for **System B** than for **System A**.

The hardware factors of both computing systems effects the performance of SpMV in two major ways – overall execution speed and impact of data locality. The GPU device of **System B** uses a *double-pumped* logical graphics pipeline that is expressed in hardware as extra context switch chip; so twice the data input per clock cycle is massaged by the larger number of processing cores of **System B** over that of **System A**. This performance difference is seen from

the data plots defined as Figure 19 and Figure 20 where the later executes at approximately half the time as the former.

As stated previously in section 3.3 of this chapter, the mitigation of locality issues derived via the employment of BCSR2x2 data compression has a lower effect on performance for **System B** than for **System A** due to the presence of a hardware-level cache on **System B** that does not exist on **System A** [59-61]. An interesting artifact of the hardware, software, and algorithmic interplay can be seen when increasing the number of threads per block for **System A** and **System B** as shown in Figure 19 and Figure 20 respectively. The general performance in both cases is similar but shifted to the right for **System B**, e.g. address coalescing appears markedly improved at 32 threads per row for **System A** and 64 for **System B** – this differential is likely an artifact of memory device I/O.

CUDA defines the unit of execution to be a *warp* which is a collection of 32 threads working simultaneously - coalescing memory addresses within this grouping [37, 62]. The memory device I/O employed by **System B** can execute twice for a single clock cycle [59-61] e.g. 32-bits per every 2-cycles means 16-bits for a single-clock cycle - each floating-point operation requires at least 32-bits as per the data-type; so the 32-bits metric can be extended as 32-threads. **System A** employees a memory device I/O with single clock cycle execution, effectively creating a 16-to-32 comparison, thus **System B** will coalesce at double that of **System A**, i.e. 64 threads versus 32 threads. Related to the concept of memory address coalescing is shared memory banks.

Shared memory is a software managed *cache-like* structure, heavily banked to align with the **Single Instruction Multiple Data** (SIMD) *lane* width of the processing core – as with address coalescing, proximity of these banks to thread accesses is important [25, 37]. These banks, sometimes called segments, execute optimally with address interleaving such that given float *pointer fp* in bank *B* and % being defined as the modulus operator, *fp* + 1 points to the address at and $(B+1)\%16$ and $(B+1)\%32$ for **System A** and **System B** respectively - each bank holding a 4-byte access per cycle [37, 77]. Critical to performance using shared memory is the avoidance of *bank conflicts* which can present any time data access is not sequential. A *bank conflict* occurs when more than one memory access is made to the same bank in the same clock cycle – successive 32-bit words are shared among 16 banks for **System A** and 32 banks for **System B** [54, 61, 62, 69, 74, 77]. CUDA handles a bank conflict by serializing each of the contending threads, for example: given *N* memory accesses and *N* unique shared memory banks

bandwidth is increased by a factor of *N* with no conflicts but is decreased by $\frac{1}{K}$ for each *K* thread that requires serialization [37, 62, 77]. Figure 21 illustrates a bank conflict on a generic CUDA GPU device.

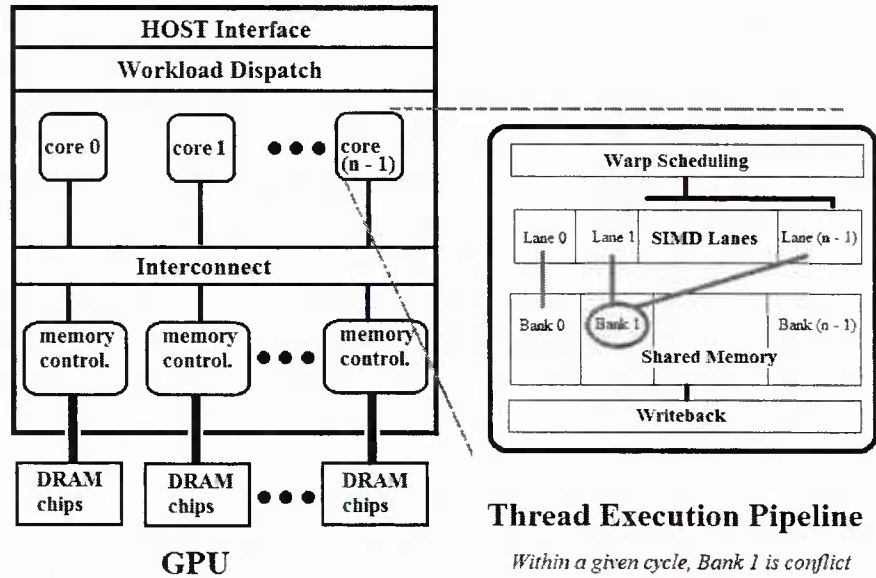


Figure 21. Block diagram of memory bank conflict for generic CUDA device

Regardless of the individual factors discussed, both CPU/GPU computing systems analyzed expose performance boosts for SpMV – as shown in Figure 22.

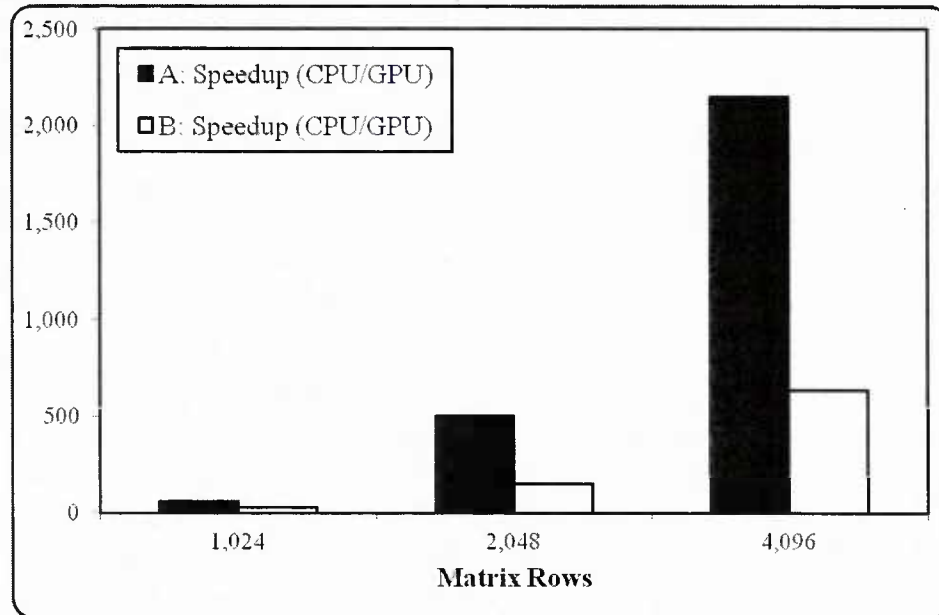


Figure 22. System A and System B Speedup factors for execution of SpMV operation

This chapter has illustrated that execution of SpMV in the CPU/GPU computing system, the highest cost of the PCG iterative solver, displays impressive improvement over the CPU-only version. The factors of software, hardware, and algorithm have demonstrated inter-dependency

in regards to the resulting performance of SpMV – how these factors affect the full candidate solution for composite process flow modeling analysis is discussed in the next chapter.

CHAPTER 4

Full Candidate Application – Single CPU/GPU Computing System

This chapter focuses on the full solution to the candidate composite process flow modeling application within the context of a *single* CPU/GPU computing system for both **System A** and **System B**. The full solution of the computationally intensive candidate application is mapped to the CPU/GPU computing system and validated against an analytical solution for all computing systems involved. The validated application is then executed using **System A** and **System B** and the resulting performance is analyzed to determine how the hardware and software factors work together to impact the resulting application computational performance. During the mapping, key computationally intensive kernels are presented and associated GPU developments explored.

This chapter will ascertain how the hardware architectures of **System A** and **System B** work together with the software factors to denote the application performance – key in this discussion is the calculation of a computational complexity analysis. The computational complexity analysis is actualized as a performance modeling equation that can be used to project how different problem, software, and hardware parameters will affect performance. Understanding these variations in factors/parameters is essential as new computing architectures arrive to get optimal performance for HPC applications in many legacy and new computational modeling codes / code developments.

4.1 Mapping Full Candidate Application to GPU

The mapping of the full candidate application to the single CPU/GPU environment is a natural extension from the previous chapter, detailing the mapping of the SpMV operation, in the *single* CPU/GPU environment for both **System A** and **System B**. Both of these mappings are done within the single shared address space CPU/GPU computing architectures and the SpMV is the largest component of the PCG iterative solver [48-50] and hence representative of the full solution itself. The presented candidate application employees the **Concurrent Number Cruncher** (CNC) GPU solvers package by Luc Buatois, et al [50] which uses a custom SpMV operation with CSR and BCSR2x2 data compression formats as well as Nvidia's **CUBLAS** library [78] for SAXPY and DOT-PRODUCT calls [50] (see Figure 23). Nvidia has recently released a library for sparse matrix operations known as **CUSPARSE** with restrictions to CUDA Compute Architecture of *at least* Version 1.1 [79]. **System A** falls into the CUDA Compute Architecture of 1.0, therefore in an effort to apply consistency across CPU/GPU computing systems the **CUSPARSE** library was not used in this study.

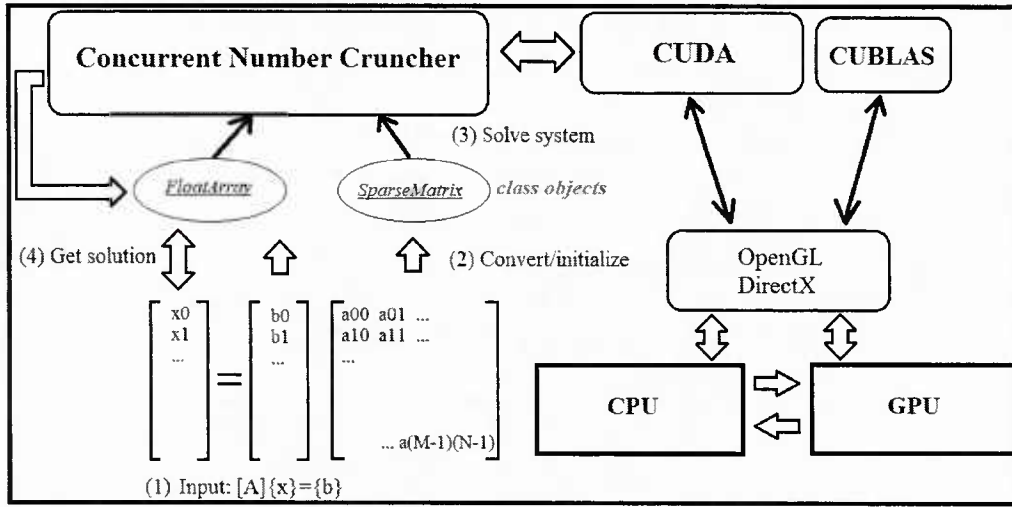


Figure 23. The CNC operational flow from input to output.

The key computationally intensive Kernels presented in the CNC software as well as the process of mapping the presented candidate application to function in a CPU/GPU computing system are discussed next.

4.1.1 Key Computationally Intensive Kernels. The most computationally intensive kernels of the full candidate solution are those that relate to the iterative PCG solver, as it is called $(K \times L)$ times with K the number of iterations for mass-convergence and L the number iterations for all nodes to be determined as *filled* (see **Algorithm 2.1**), as well as the number of CG iterations for each solution call to the linear equation solver. The SpMV is the largest cost of the PCG iterative solver and is ported to the local GPU device, as with the SAXPY and DOT-PRODUCT kernels, and called within the *nested loop* described in **Algorithm 2.1**, but the consideration of GPU code developments, software, and hardware factors is critical for optimal performance results.

4.1.2 GPU Code Developments. The PCG iterative solver defined in the candidate application is mapped and ported to GPU via the CNC code as shown in Figure 24. The CNC code is third-party software created using the C/C++ programming language that embeds the CUDA Kernels within its design – the library header for the CNC package is simply placed as part of the preprocessor “include” calls and compiled with the proper CUDA library links using the NVCC compiler and an executable is created with separate GPU-bound code to leverage the local GPU device [37, 38, 50]. The CNC package solves the system of linear equations of the matrix form $\underline{Ax} = \underline{b}$ receiving input matrix data from the calling C/C++ class file(s), executing the GPU device code, and retrieving the resulting solution vector for the CPU-bound code portion of the application [50]. The CNC package defines the PCG iterative solver, as with all

PCG solvers [46], using multiple calls to the SpMV, SAXPY, and DOT-PRODUCT operations – the SAXPY and DOT-PRODUCT portions of the solver are examined next.

4.1.2.1 CUDA Kernels of PCG solver The SAXPY and DOT-PRODUCT operations are defined in the Nvidia **CUBLAS** library [78] which is the CUDA version of the Basic Linear Algebra Subprograms (BLAS). BLAS is categorized by levels 1, 2, and 3 with level-1 consisting of vector operations, level-2 consisting of matrix-vector operations, and level-3 consisting of matrix-matrix operations [78, 80, 81]. **CUBLAS** follows the BLAS paradigm, defining the SAXPY and DOT-PRODUCT as level-1 category operations [78].

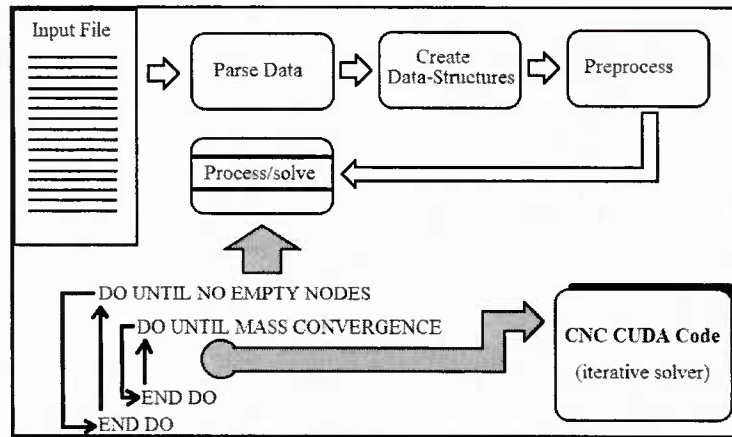


Figure 24. Presented candidate application diagramming the placement of CNC Package.

There are several steps that must be followed for any CPU/GPU computing system to use a CUDA library package which can be generalized as *initialization*, *allocation*, *execution* and finally *retrieval* of results [37, 62]. Initialization is the first step and is potentially heavy due in no small part to the sheer size of the library itself. The **CUBLAS** library is quite large, containing the **Shader Assembly** (SASS) and PTX code for every Kernel defined in the library with PTX as much as 75% of the library size [37, 38, 62]. The SASS is the binary version of the library and PTX the intermediate to allow for differing GPU device generations and is loaded using a **Just In Time** (JIT) compiler construct – the driver has to locate and read the SASS binary for the particular GPU device and load it to the machine's board. Once the library is loaded, the CPU issues a command for the GPU device to allocate memory for the number of elements and the data-type that is to be used by the Kernel. The data is then passed to the GPU from the CPU via pointers as direct contact between CPU and GPU is not possible [24, 37, 62]. The kernel is executed from input data and passed back to the CPU host for use by the system – the GPU memory is not explicitly removed until another call to the allocation is made, the GPU is a state-machine by design [8, 31, 34, 35].

The **CUBLAS** scalar DOT-PRODUCT function executes a single memory access for every arithmetic operation yielding an upper bound on performance that becomes the ratio of CPU to GPU memory sizes, typically ranging around 5 to 10 [37, 78, 82]. The cost of access to the data residing in CPU space erodes any performance boost. This is why the full implementation of the candidate application keeps data located in GPU space [50] – instead of multiple global accesses during the iterative phases of the solver, single calls are made at the beginning and end of the solver [50]. The computational cost of the scalar DOT-PRODUCT is negligible relative to the SpMV but its role in the PCG algorithm is invaluable as these computations are used to define the solution convergence of the linear equation system [46, 65].

The scalar DOT-PRODUCT is primary in the computation of residuals [46], applied in **Algorithm 2.2** at lines 11 and 12. The closer the residuals are to zero, the more likely a solution has been located – however this assumption could become tenuous if the device employed is not fully compliant with IEEE-754 floating-point representation standards [83]. The potential for non-compliant floating-point operations using GPU devices is a valid concern [84] as GPU manufacturers have never openly held to this compliance and this was never a concern [83, 85, 86]. The typical high-res games redraw frames at up to 60-times a second [2, 24] so any visual artifacts produced by a slightly-off floating-point value would not be noticed even by the most perceptive human player. Once the GPU moved into the exacting analytics required of the engineering/scientific community, the formerly lax adherence to numerical accuracy standards needed to be tightened. The algebraic representation of the scalar DOT-PRODUCT operation is shown in equation (4.1) with \underline{A} and \underline{B} vectors each of length n .

$$\underline{A} \bullet \underline{B} \equiv \sum_{i=1}^n A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n = \underline{AB}^T \quad (4.1)$$

Nvidia has long maintained that its GPU devices held fast to the IEEE standards but admitted that the more accurate double-precision representation was not supported [86], at least until the advent of the Tesla Fermi architectural designs [60, 61]. These architectures are not only double-precision compliant they contain Error Correcting Code (ECC) adaptations to avoid propagation of small numerical inaccuracies [59].

Perhaps counter-intuitively, the real instantiation of floating-point inaccuracies involving the scalar DOT-PRODUCT is the addition portions of the operation rather than the multiplication [13, 45, 87]. The reason is the potential for alignment error when normalizing the result to be 127-biased [13, 87]. Figure 25 shows an example conversion from a base-10 floating-point number to the corresponding base-2 representation with applied 127-bias.

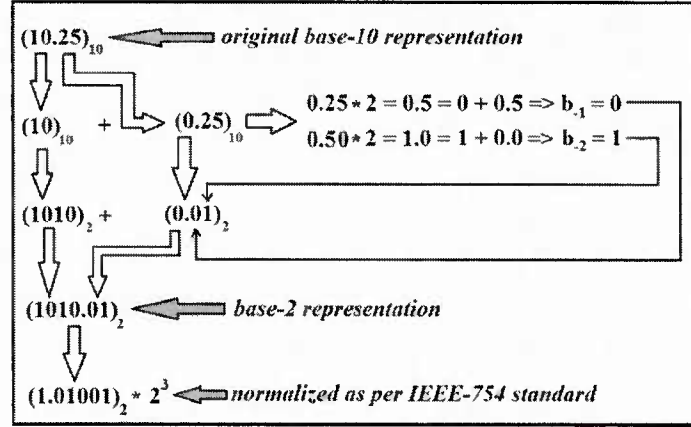


Figure 25. IEEE conversion from base-10 to base-2 with normalization.

Unfortunately, even full compliance of IEEE standards for floating-point representations is no guarantee of full accuracy in every case [45, 87].

Paraphrasing work by Anthony M. Castaldo [87], given two numerals with exponents e_1, e_2 such that $e_1 > e_2$ the number of bits to exceed the scale of the returned value is $e_1 - e_2$ this is the number of zeros that the smaller numeral will be padded to on the left to properly align. Therefore, if the value of smaller operand is just one greater than the bits of the mantissa it adds *nothing* to the resulting addition. Supposing the 32-bit format, if the aforementioned exponents differ by 24 or more, the answer will be the larger of the two – the other operand is completely ignored in the result. The operation still ascribes to the IEEE-754 standard *but* is completely erroneous with regards to small numerical variations in the long term; particularly sensitive to these perturbations are large clustered machines where the error can magnify as it progresses through the system, known as a *soft error* [84] – hence contributing to the importance of ECC adaptations.

The **CUBLAS** SAXPY operation executes a scalar multiplication and addition with vectors as shown in equation (4.2) with y and x vectors and α the scalar. The **CUBLAS** version of this level-1 BLAS operation is generalized to allow for incremental steps in both x and y directions, i.e. an array stride [78, 88]. These extra levels of indexing could cause a slight drop in performance but is a trade-off to the generality provided by the library. The SpMV operation is the same used in the previous chapter – complete with the same determination of optimal thread occupancy.

$$y \leftarrow \alpha x + y \quad (4.2)$$

The full candidate application is validated against an analytically derived solution for a simple 2D unstructured mesh model consisting of radial center injection in a thin circular plate mold geometry using the single CPU/GPU computing system next.

4.2 Validation of Full Candidate Application on Single CPU/GPU

The correctness of the full candidate application for the single CPU/GPU computing systems is ensured via the examination of flow-front progression and injection port pressures of numerical solutions for CPU and GPU against the correspondingly analytical results. The model used for validation is simple 2D circular plate mold geometry with a center, radial injection and is compared with the resulting analytical equation.

The simple circular plate model has a radius of 10 cm and an inner radius of 0.15 cm for the radial injection port as shown in Figure 26. The inner radius, R_0 , is subjected to a constant radial flow rate Q . The thickness of the cavity is H , the injection inlet pressure is P , and is a function of transient resin infusion time, resin viscosity is μ , the permeability of the fiber preform is \bar{K} , and the porosity of fiber compaction is ϕ . The flow front radial location at any time t is given by [89]:

$$R(t) = \left(\frac{Qt}{\pi\phi H} + R_0^2 \right)^{\frac{1}{2}} \quad (4.3)$$

The corresponding expression for injection pressure, which varies with time, is given by [89]:

$$P_0 = \left(\frac{\mu Q}{2\pi K H} \ln \left(\frac{R(t)}{R_0} \right) \right) \quad (4.4)$$

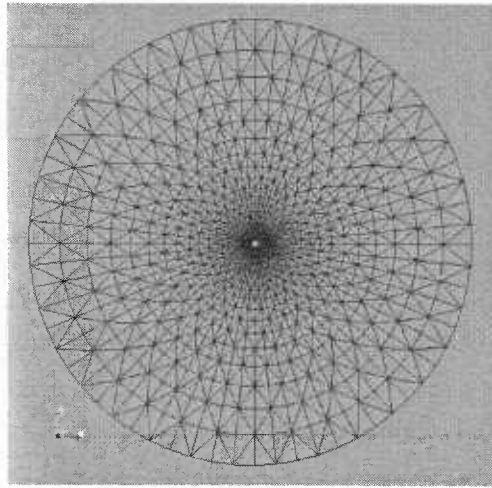


Figure 26. 2D circular plate validation model (not to scale).

The following physical parameters are used in this analysis:

$Q = 2.4 \frac{cm^3}{sec}$, permeability $\bar{K} = 44.0e-08 cm^2$, a viscosity $\mu = 0.02 PaS$, a porosity of $\phi = 0.805$, a time step $\Delta t = 0.5$, and an element thickness $H = 0.742 cm$. The circular plate model involved a computational mesh of 1,344 nodes and 2,560 3-noded triangular elements. Figure 27 and Figure 29 display the flow-front progression showing the computed and analytical variation of the radial flow front location with respect to time for **System A** and **System B** respectively and clearly define accuracy with the analytical solution. Figure 28 and Figure 30 display the corresponding transient inlet injection pressure for **System A** and **System B** respectively and clearly define accuracy of the computational solution.

The flow-front and inlet injection pressure values are accurate for this circular plate radial injection geometry. Other complex flow modeling geometries also showed equivalent comparison of the flow front progression and the predicted fill time between CPU/GPU based computational solutions for the same geometry and problem parameters employed. Computational performance modeling performance of the full candidate solution for the single CPU/GPU computing systems is presented next. The initial performance evaluations were conducted using the *single* CPU/GPU computing systems.

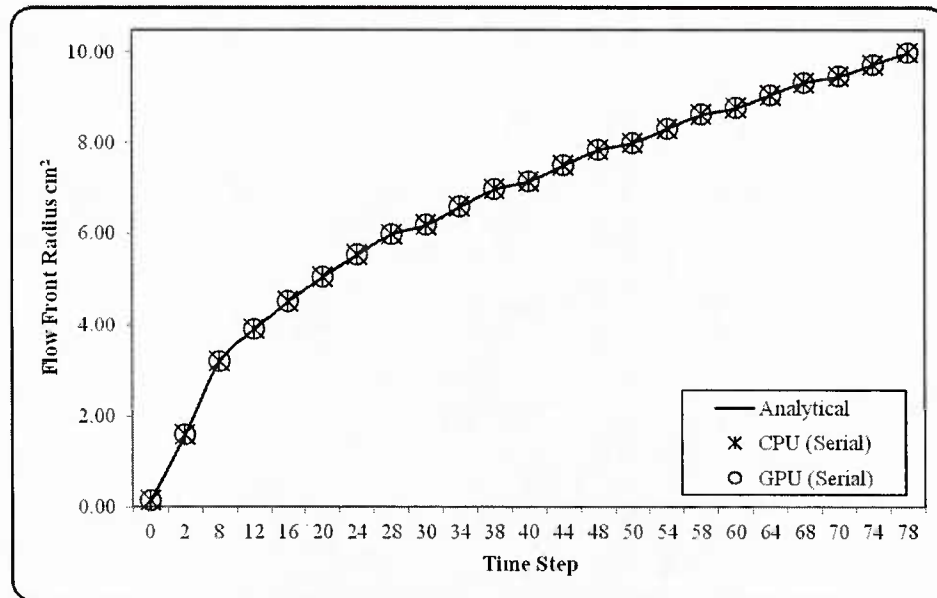


Figure 27. Validation of single CPU/GPU for flow-front progression (**System A**).

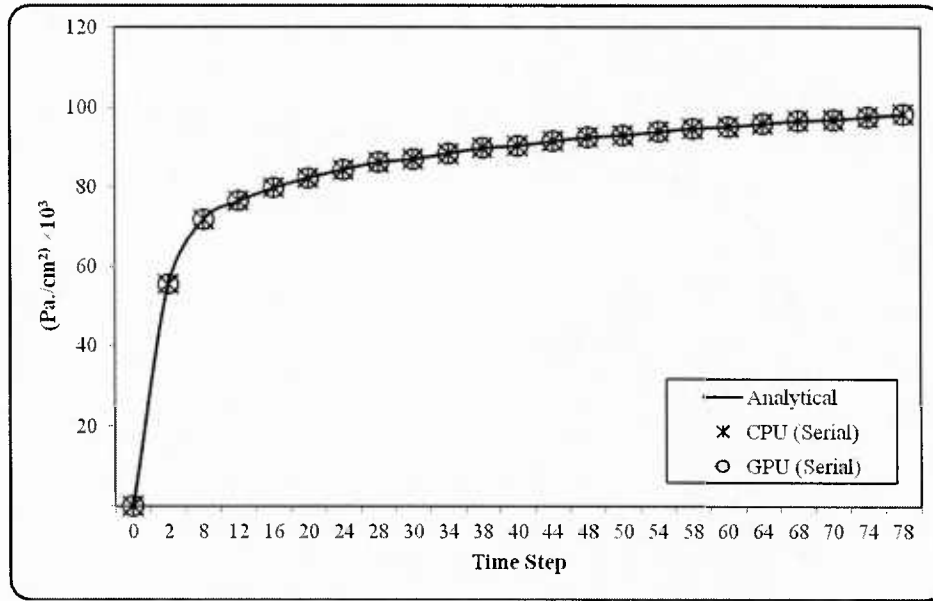


Figure 28. Validation of single CPU/GPU for inlet injection pressure (**System A**).

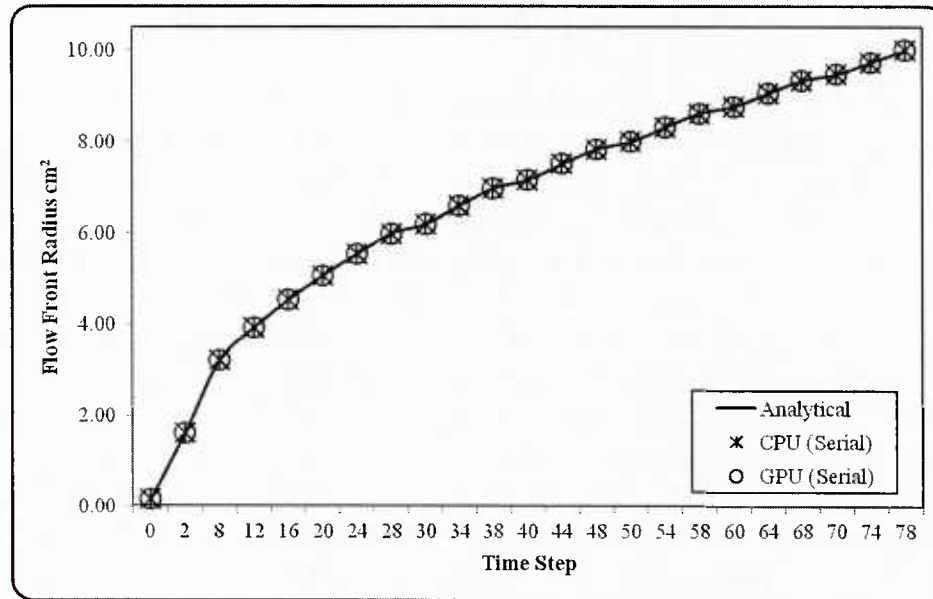


Figure 29. Validation of single CPU/GPU for flow-front progression (**System B**).

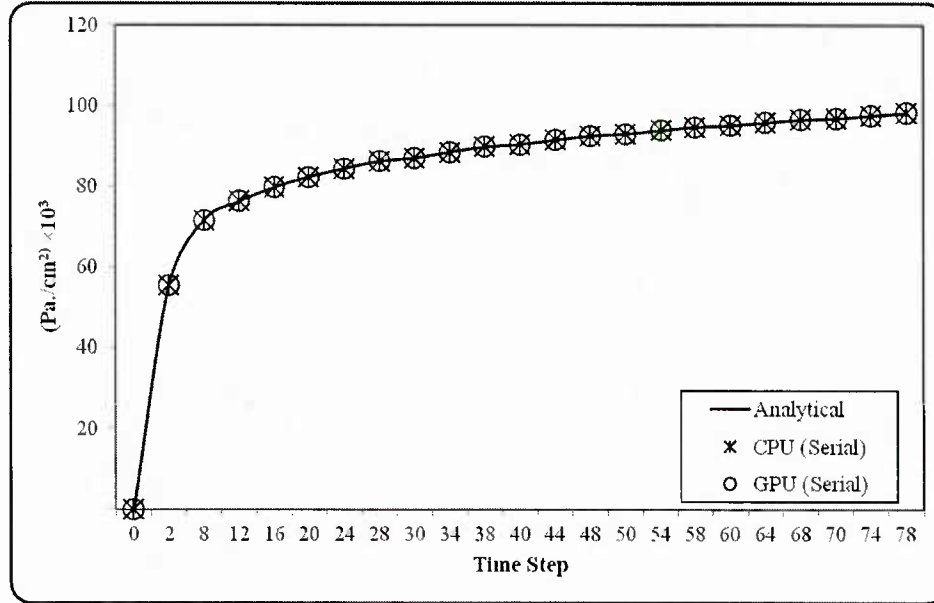


Figure 30. Validation of single CPU/GPU for inlet injection pressure (**System B**).

4.3 Initial Full Candidate Application Performance on Single CPU/GPU

The results of full candidate application on the *single* CPU/GPU for both computing system machines, **System A** and **System B**, with unstructured meshes **MA**, **MB** and **10FT** as input model data and the CSR data compression format are presented in this section with the goal of exposing the performance effects of software, hardware, and algorithmic factors using a consistent model in differing computing environments. Clearly, the most salient metric of performance measures for scientific and engineering applications where analysis time is concerned is the direct measure of computational solution time. However, due to the inherent processor differences in terms of clock speeds, benchmarking different hardware is difficult to quantify. Total computational time employed also depends on the cost of arithmetic operations in different architectures that can vary, and detailed information in commodity processors used in this work are proprietary [66, 72, 88]. A comparison of the **Floating-Point Operations** rate (FLOPs) may also be misleading in diversified architectures with different clock frequencies – therefore FLOPs are normalized by the clock rate as defined in section 3.2 of chapter 3. Normalizing FLOPs helps to avoid idiosyncrasies of individual hardware, to provide arithmetic power comparisons. Execution time depends on processor speeds and normalized FLOPs allow for a more definitive comparison over wall-clock time or FLOPs.

The *single* CPU/GPU computing system defined as **System A** is examined first followed by **System B**.

4.3.1 System A. The *single* CPU/GPU computing system execution of the full candidate application was compared against the CPU-only version. *Table 9* shows that the GPU outperforms the CPU-only system for process flow modeling analysis employing input mesh **MA**, **MB**, and **10FT** when examined with regards to total solution wall-clock time. Both flow modeling analysis obtained same flow progression contours and predicted infusion time for the same physical problem parameters employed in all cases. The superior arithmetic power of the GPU for this *single* CPU/GPU computing system is clearly visible in *Table 10*, presenting an advantage of more than 9-times the number of KFLOPs, which have been normalized as per section 3.2 of chapter 3, in the GPU compared to the CPU.

Table 10

Full solution performance with single CPU/GPU and CSR compression (System A)

Unstructured Mesh	CPU Time (secs.)	GPU Time (secs.)	Speedup Factor
MA	6,176.46	418.44	14.76
MB	81,929.7	4,219.61	19.42
10FT	6770.31	285.17	23.74

Table 11

Full solution KFLOPs with single CPU/GPU and CSR compression (System A)

Unstructured Mesh	KFLOPs (CPU)	KFLOPs (GPU)
MA	142.26	1,169.56
MB	133.39	1,292.44
10FT	135.92	1,186.73

4.3.2 System B. The *single* CPU/GPU computing system execution of the full candidate application was compared against the CPU-only version. *Table 11* shows that the GPU outperforms the CPU-only system for input mesh **MA**, **MB**, and **10FT** when examined with regards to total solution wall-clock time. The superior arithmetic power of the GPU for this *single* CPU/GPU computing system is clearly visible in *Table 12*, presenting an advantage of almost 3-times the number of KFLOPs, normalized as per section 3.2 of chapter 3, for the GPU compared to the CPU.

Table 12

Full solution performance with single CPU/GPU and CSR compression (System B)

Unstructured Mesh	CPU Time (secs.)	GPU Time (secs.)	Speedup Factor
MA	424.93	168.57	2.52
MB	6,414.74	1,197.35	5.36
10FT	627.83	105.09	5.97

Table 13

Full solution KFLOPs with single CPU/GPU and CSR compression (System B)

Unstructured Mesh	KFLOPs (CPU)	KFLOPs (GPU)
MA	2,299.38	3,213.32
MB	1,891.28	5,046.01
10FT	2,142.75	4,743.81

4.3.3 Initial performance analysis. The execution of the unstructured mesh input files, **MA**, **MB**, and **10FT** for both computing systems exposes some common performance behaviors – most notably the correlation to problem size and performance. **System A** and **System B** produce better performance using total solution time and normalized FLOPs as metrics when the problem size increased. This performance boost for increasing problem size is reflected in the previous chapter’s analysis of the SpMV for both CPU/GPU computing systems as well as though out the published literature regarding GPU performance [2, 25, 67, 90]. However, there are some differences with the magnitude of the performance boost found. **System B** has a lower speedup factor and KFLOPs change than does **System A**. Likely this is due not to any defect of **System B** but rather the more advanced CPU used – the dual-core CPU of **System A** is so lacking in relation to the GPU that the speedup factor must be greater.

4.4 Software Data-Structures/Layouts Factors

This previous section was an initial performance analysis for three unstructured mesh model inputs via the *single* CPU/GPU computing systems defined as **System A** and **System B** and produced some good performance boosts, especially for **System A**. However, the initial software variables need to be examined to identify potential factors that can hinder performance of the presented candidate application. The first software factor to be examined is data compression format.

The initial performance was executed using CSR, a data compression format with a noted proclivity for poor computational intensive performance [53, 54] – this has been shown to improve with BCSR2x2 due to increased locality [52-54]. The presented candidate application generates systems of equations based on 3-noded triangular elements each with 1-degree of freedom resulting in dense sets of sub-blocks in the sparse matrix [44, 45] – a potential boon for locality using BCSR2x2.

The presented candidate application generates dense sub-blocks of non-zero elements via the methodology of the **Finite Element Method** (FEM) - initial collections of 3x3 local element matrices result from 1-degree of freedom applied to the input 3-noded triangular elements [44, 45]. These local element matrices are coalesced into a global element matrix that maintains symmetry from these 3x3 sub-matrices [44, 45] – the 2x2 sub-blocks utilized by the

BCSR2x2 data compression will be subsumed by the dense sub-matrices of the global element matrix. The application of data compression formats to improve locality is closely associated to memory address coalescing and thread occupancy as all of these seek to maximize throughput by grouping as many threads as possible, however the CNC GPU solver requires multiple Kernels which forces a schism to these groupings.

CUDA places implicit barriers between dependent Kernel invocations [37], e.g. Kernels that rely on one or more currently executing Kernels must wait for system synchronization to occur before continuing [62, 65]. This new software factor subsists with the PCG iterative solver because it is composed of not just the SpMV Kernel but a number of **CUBLAS** calls executed in sequence. **System B**, as part of the CUDA Compute Architecture 2.0, can simultaneously apply Kernels using the CUDA *stream* [60, 61] but this is not a luxury offered by **System A** [62, 76]. Therefore, to maintain as much consistency of variables between systems, these *stream* constructs were not employed in the present work. The role of the data compression format is discussed next.

Figure 31 and Figure 32 show the execution time for **System A** and **System B** respectively revealing a distinct difference in the performance benefit of changing from CSR to BCSR2x2 for each of these CPU/GPU computing systems. Figure 31 shows the performance boost of increased locality for the presented candidate application expressed as the BCSR2x2 data format for input meshes **MA** and **MB** with negligible effect for the less regular **10FT** mesh configuration whereas Figure 32 describes a deleterious result for meshes **MA** and **MB** – again **10FT** is negligibly affected. This difference in performance is surprising from a pure locality approach as the dense sub-blocks created by the 3-noded triangular elements are the same for both **System A** and **System B** – the arithmetic power leveraged by the different CPU/GPU computing systems via the normalized FLOPs is shown in Figure 33 and Figure 34 for **System A** and **System B** respectively.

System A shows a much higher rate of floating-point operations when going from CSR to BCSR2x2 than **System B** for the corresponding change (see Figure 33 and Figure 34) – excepting the **10FT** model mesh configuration which illustrates negligible difference in both computing environments. **System B** manifests a negligible increase in normalized FLOPs for the BCSR2x2 format and a decrease in performance – given the relative regularity of the input mesh configuration; this implies an initially poorer locality for **System A**. The lower impact of the locality change for **System B** is likely due to the implementation of an actual cache as per the Nvidia *Fermi* architecture [60, 61]. The negligible change for the **10FT** input mesh supports a correlation not just to the size of the problem domain but the expressed *regularity* of the matrix. The hardware factors on performance for both **System A** and **System B** are discussed next.

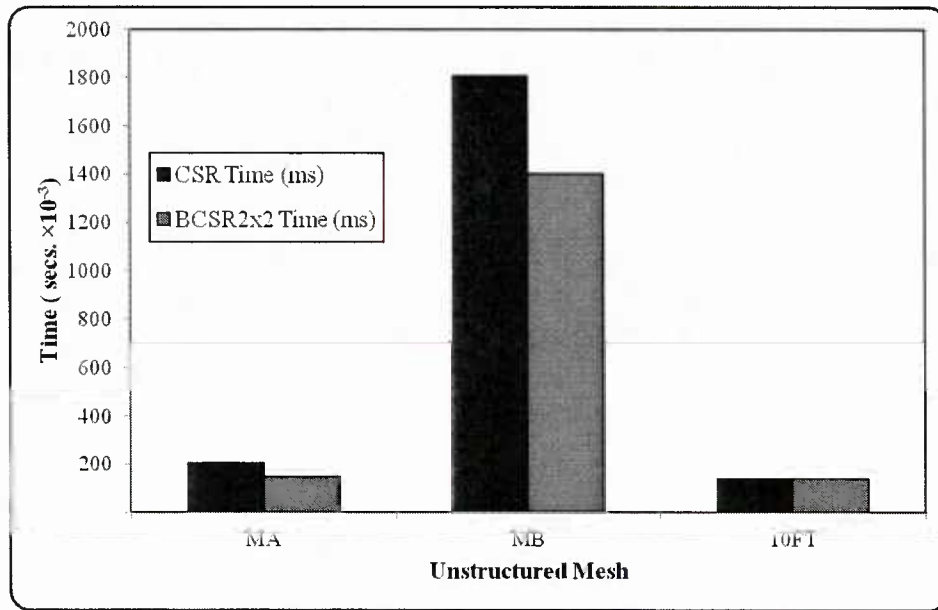


Figure 31. Full candidate solution performance (System A).

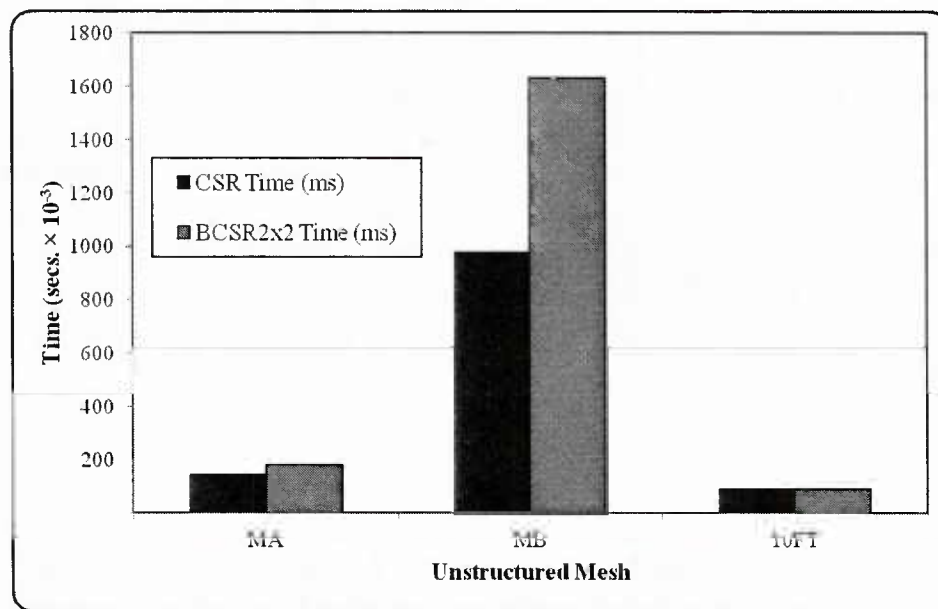


Figure 32. Full candidate solution performance (System B).

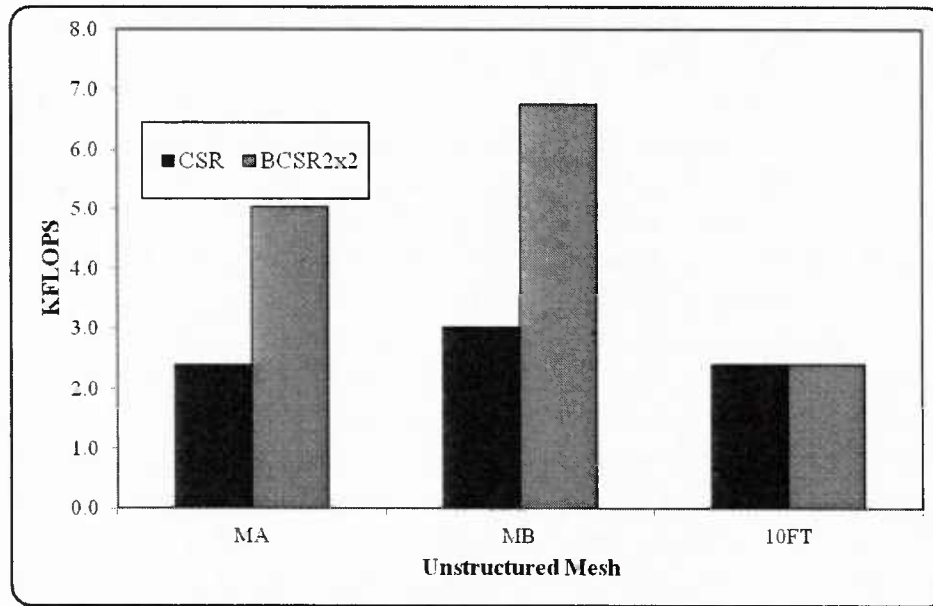


Figure 33. Full candidate solution KFLOPs performance (System A).

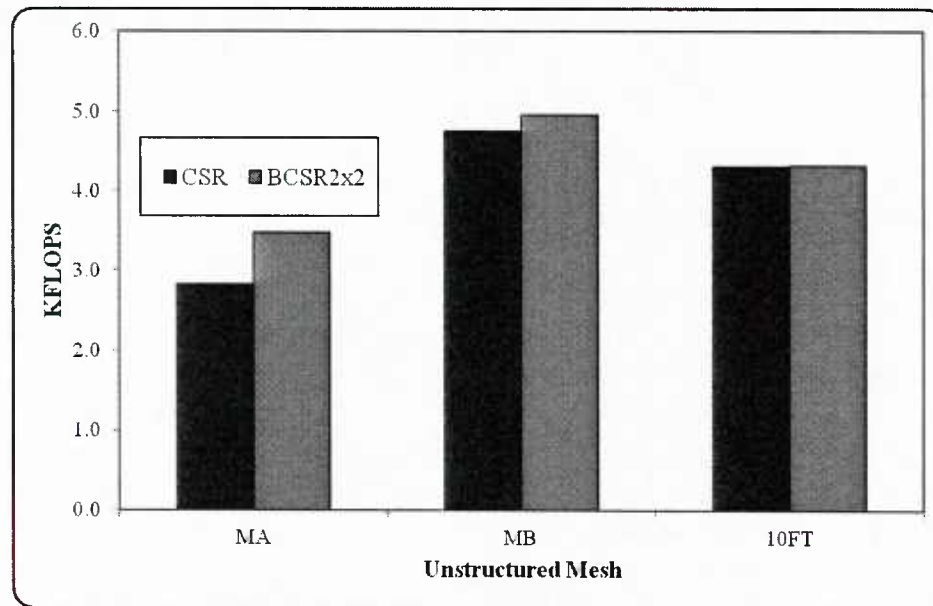


Figure 34. Full candidate solution KFLOPs performance (System B).

4.5 Hardware Architectural Factors

The observed performance of the CPU/GPU computing systems using the unstructured mesh configuration defined by **MA**, **MB**, and **10FT** is affected not just by the software factors discussed in the previous section but hardware factors as well. The presented candidate

application resulted in a performance boost for both **System A** and **System B** when using the CSR format albeit to a smaller degree for **System B** – excepting the negligible results observed for the less regular input mesh **10FT** in both computing environments. However, when the BCSR2x2 format was employed the performance of **System B** dropped while **System A** increased – fairly regular geometries, consistent models and software factors were used leaving the underlying hardware architectural factors as culpable.

System B has a cache defined at the hardware level [60, 61] whereas **System A** does not [58, 76] – this has great impact on solutions involving sparse matrices such as those applied via the presented candidate application [54]. A *hardware-level cache* provides the ability to stash a process during execution and quickly retrieve it when needed rather than calling global memory read/writes at every instant. The more advanced architecture of **System B** does not negate the importance of judicious application of software factors, but it does alleviate it somewhat. The lowered importance of locality on **System B**, expressed using BCSR2x2, adds all the computational overhead of extra loops to iterate over defined sub-blocks but none of the corresponding utilization of greater arithmetic units and hence none of the predicted performance seen in **System A**.

The software and hardware factors encountered during the execution of computationally intensive applications using CPU/GPU computing systems are typically difficult to quantify and as such defining concrete metrics for performance on these system is difficult. The formulation of an expected performance equation is discussed in the next section.

4.6 Computational Complexity Analysis

Computational complexity has historically been quantified using asymptotic analysis to understand how the design scales [91], but this methodology relies on axioms that do not exist for GPU computing such as constant operations yielding negligible costs to overall performance [92]. Adjusting the number of threads per block can allow CUDA to coalesce memory addresses resulting in as much as 20-times reduction in execution time [72] – *small* changes can have *big* impacts on the algorithm [66]. Therefore quantifying algorithmic behavior is relegated to the minutia of performance modeling as standard parallel modeling techniques fail to take into account the importance of limited and high access costs to the *exposed* memory hierarchy of the modern GPU [16, 72].

Currently there is no standard for performance analysis for use with GPGPU computing, however options have been published [16, 66, 67, 72, 93] and this study follows many of the concepts put forth in these works. The code that is ported to the GPU(s) for this research is broken up into its major components, analyzed, and relevant parts re-assembled to form a

general performance model. The *single* CPU/GPU computing system is studied first and expands to the *multiple* CPU/GPU computing system in the next chapter.

The following sub-section will derive a mathematical model for a *single call* to the PCG iterative solver rather than the whole solution and is detailed in Appendix A. This approach yields a base equation from which the different behaviors for different parameters can be determined. The third party CNC software was used for the results being analyzed which defines 16×16 thread blocks [50] – this is the assumed constant for all derived equations.

4.6.1 Single call to PCG solver. Computing the *average number of non-zero* elements for a given sparse matrix that is assumed to be square with 16×16 thread blocks, R_{NZ} with G_{mem} , M are the number of *bytes* in global memory and the total number of rows respectively – using *bytes* as the metric defines the numerical values of 4 given that there are 4 bytes for every floating-point data type expressed in equation (4.5).

$$R_{NZ} \cong MAX \left\{ 1, G_{mem} - \frac{4 \times (M + 1)}{(4 + 4) \times M} \right\} \quad (4.5)$$

Computing the *number of blocks*, N_B , with N_t , N_w , and N_{smp} the number of threads per warp, the number of warps per block, and the total number of symmetric multiprocessors for M rows and average non-zero entries per row as R_{NZ} columns can be defined by equation (4.6).

$$N_B \equiv \frac{M \times R_{NZ}}{N_t \times N_w \times N_{smp}} \quad (4.6)$$

Computing the *cycle cost* based on the defined cost of sparse matrix-vector computational costs for the serial CPU at $2 \times (N_{rows} \times N)$ [46, 91] and using a global memory cycle count of 500 based on the average for the actual data range of 400 – 600 is C_T , with B the number of threads per block expressed in equation (4.7).

$$C_T \equiv 2 \times \left(\frac{500 + M}{\sqrt{B}} \right) \quad (4.7)$$

Computing the total cost for execution of preconditioned conjugate-gradient solver on GPU, C_{pcg} with K , R and D the total number of iterations for convergence, memory clock frequency, and number of cycles per pipeline can be defined by equation (4.8).

$$C_{pcg} \cong K \times \left(N_B \times N_{smp} \times C_T \times \frac{1}{D \times R} \right) \quad (4.8)$$

Equation (4.8) is not exact but does reflect many of the same performance modeling equations in published works [66, 72].

Figure 35 and Figure 36 depict a comparison of actual to estimated performance time for **System A** and **System B** respectively using unstructured mesh configuration **MA**, **MB**, and

10FT expressed with the CSR data compression format. The results are not exact, as the thread dispatch policy for both GPU devices is *non-public* information [72, 93] and is *interleaved* for optimal throughput [2, 37, 55]. In addition, synchronization costs between dependent Kernels of the PCG iterative solver are not published. The issue is further compounded with an assumed not large but with an unknown dispatch policy. The approximate nature of the results provided by the derived performance model demands the examination of normalized error which is computed by $\frac{|A_t - E_t|}{A_t}$ with A_t and E_t defining the actual and estimated times respectively.

Figure 37 and Figure 38 show the normalized error of the single PCG call for **System A** and **System B** respectively and indicate values of less than 50% which is a manageable amount of error for the given input data – the data results are shown in *Table 14* for **System A** and *Table 15* for **System B**. The next section extends from the single call to the PCG iterative solver and mathematically models the *full candidate solution*.

Table 14

GPU device limits for both systems

System A: Compute Architecture 1.0	System B: Compute Architecture 2.0
Core Clock Rate = 1.35 GHz	Core Clock Rate = 1.15 GHz
Total Warps = 24	Total Warps = 48
Total SMP = 16	Total SMP = 14
Shared Memory per Block = 1024	Shared Memory per Block = 1024
Registers per Block = 512	Registers per Block = 256
Shared Memory Banks = 16	Shared Memory Banks = 32

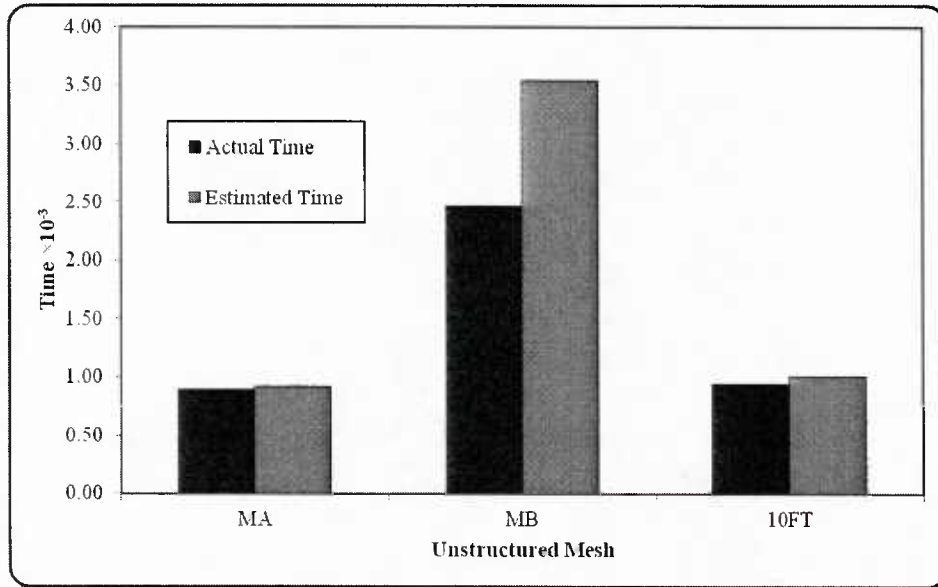


Figure 35. Performance modeling of single call to PCG solver (**System A**).

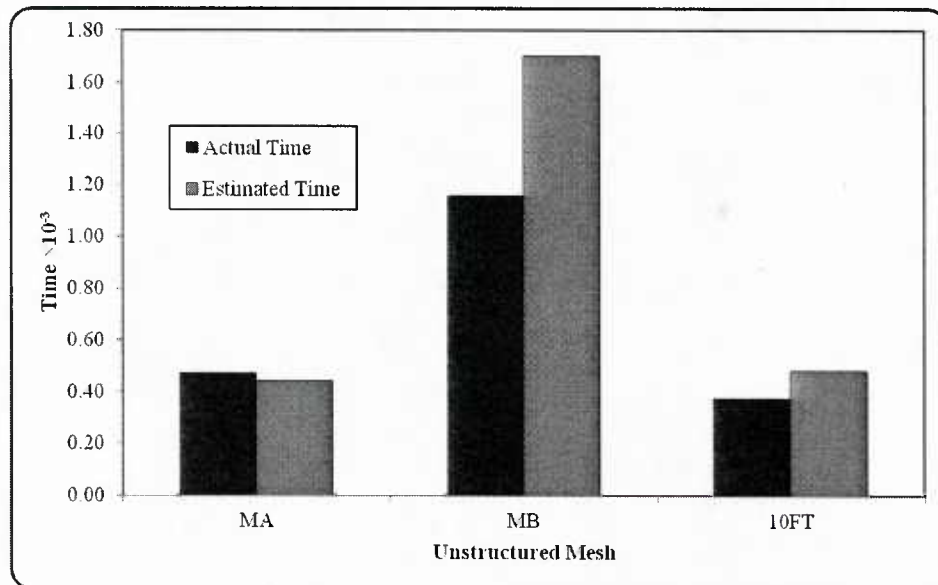


Figure 36. Performance modeling of single call to PCG solver (**System B**).

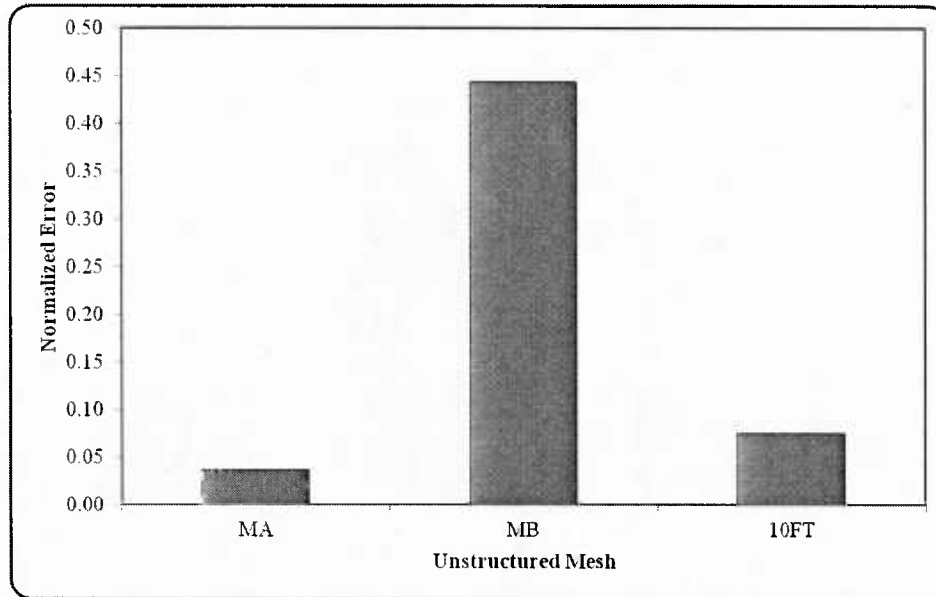


Figure 37. Normalized error for single call PCG modeled performance (System A).

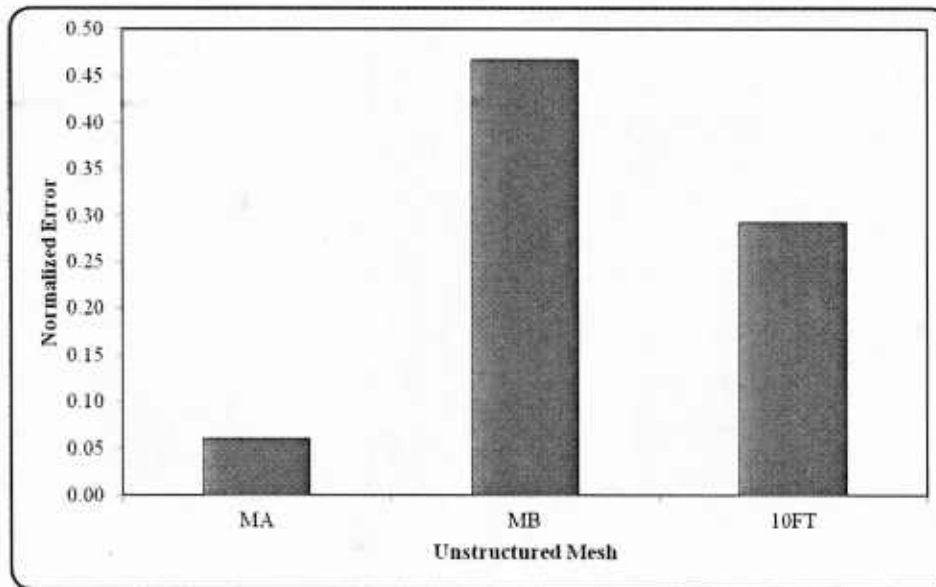


Figure 38. Normalized error for single call PCG modeled performance (System B).

Table 15

Performance modeling of single calls PCG (System A)

Unstructured Mesh	Actual Time (ms)	Estimated Time (ms)
MA	0.888224	0.920487534
MB	2.45664	3.545509018
10FT	0.938336	1.008680919

Table 16

Performance modeling of single calls PCG (System B)

Unstructured Mesh	Actual Time (ms)	Estimated Time (ms)
MA	0.470848	0.442022833
MB	1.160256	1.702796988
10FT	0.374528	0.484375628

4.6.2 Full Solution Cost – Single GPU. The final modeling performance equation for a *single* CPU/GPU computing system is derived from the estimated *single* call to the PCG computed from equation (4.8). The size and granularity of the local GPU device registers are significant to the number of active threads for utilization of computational resources and is defined by equation (4.9) with Rg_{alloc} the register allocation unit size and $W_{granular}$ the warp allocation granularity – both of which are found in hardware specifications [37, 62].

$$A_{rg} \equiv \frac{Rg_{alloc}}{W_{granular}} \quad (4.9)$$

Equation (4.9) is a major component of a defined conditional function that models the serialization of threads within the warp construct shown as equation (4.9.1) with N_{ATB} the number of active thread blocks per SMP and $N_{warp} \equiv M \times \left\lceil \frac{R_{NZ}}{N_t} \right\rceil$.

$$W_{pen} \equiv \begin{cases} IF & \frac{N_{warp} \% 4}{N_{ATB}} > 0 : \frac{N_{warp} \% 4}{N_{ATB}} \times A_{rg} \\ ELSE & 1 \end{cases} \quad (4.9.1)$$

Equation (4.9.1) is passed to the Gaussian distribution adapted to model the performance behavior of the *single* CPU/GPU computing system defined as equation (4.9.2).

$$R_{growth} \equiv \frac{1}{\sqrt{\pi \times W_{pen}}} \times e^{-\frac{1}{2}} \quad (4.9.2)$$

Computing the *total cost* for execution of solution [93] with multiple calls to the GPU-enhanced PCG, T_{gpu} with \dot{K} the total number of iterations for full solution convergence – equation (4.9.3) defines the final performance modeling equation for a single CPU/GPU computing system.

$$T_{gpu} \equiv (\dot{K} \times C_{pcg}) \times R_{growth} \quad (4.9.3)$$

Figure 39 and Figure 40 depict the results of the actual to estimated execution times for the solutions of unstructured mesh configurations **MA**, **MB**, and **10FT** for single CPU/GPU computing systems executing in **System A** and **System B** environments respectively. The formed finite element matrices for the input data meshes being solved via the presented candidate application are expressed using the CSR data compression formats in all cases.

The observed results of performance modeling for both CPU/GPU computing systems show a close equivalence of actual to estimated solution times. Both **System A** and **System B** showed dramatic decreased difference of estimated and actual time as the size of the input data increased – an inverse relationship that corroborate the premise that the underutilized and non-coalesced memory accesses can be expensive [62, 66, 72, 93]. The more intense the floating-point operations, the more fully utilized the CPU/GPU computing system resources are and the better chance of address coalescing. The increased number of input data elements being solved mitigates the impact of extraneous variables such as threaded time-sharing policies and equation (4.9.3) becomes the dominating predictor for the single CPU/GPU computing system for the full solution of the candidate application. Figure 41 and Figure 42 shows dramatic decrease in normalized error between the actual and complexity analysis predicted results for both **System A** and **System B** respectively support this precept - *Table 16* and *Table 17* are the actual results.

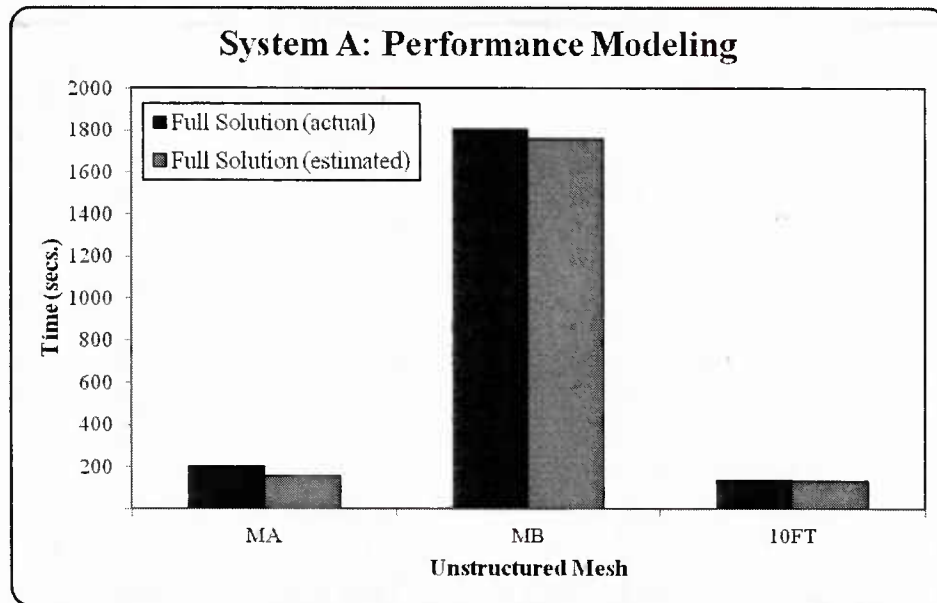


Figure 39. Performance modeling single CPU/GPU full solution (**System A**).

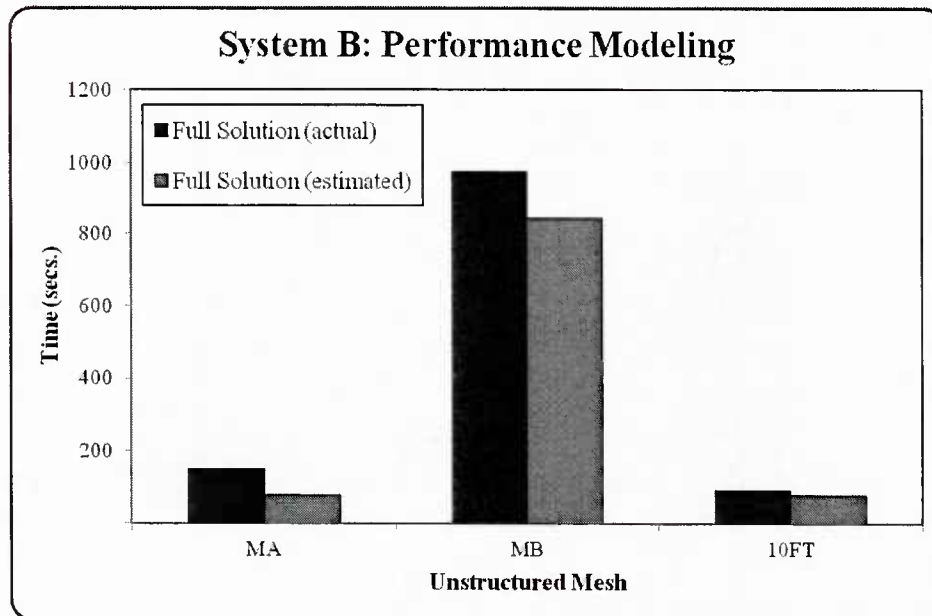


Figure 40. Performance modeling single CPU/GPU full solution (System B).

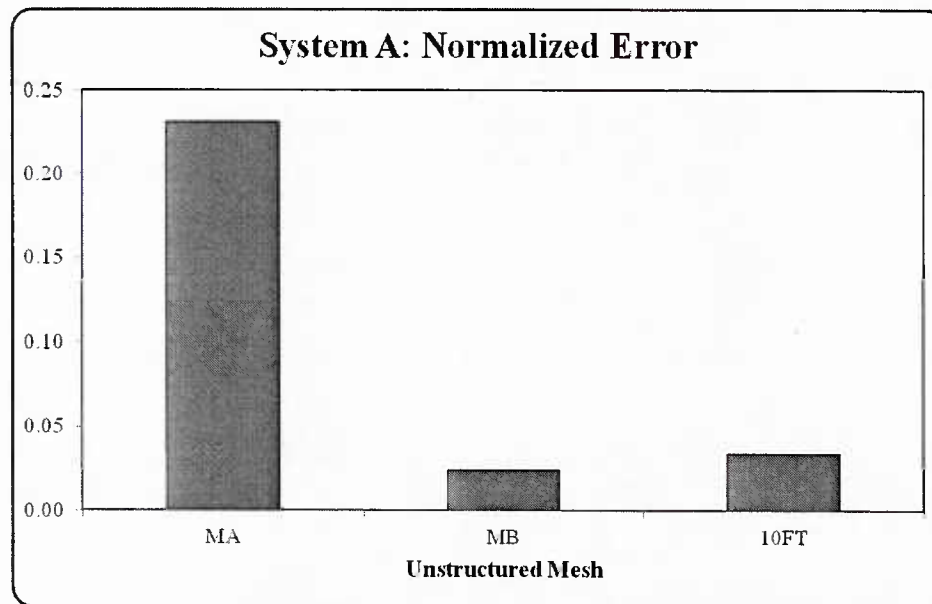


Figure 41. Error single CPU/GPU full solution modeled performance (System A).

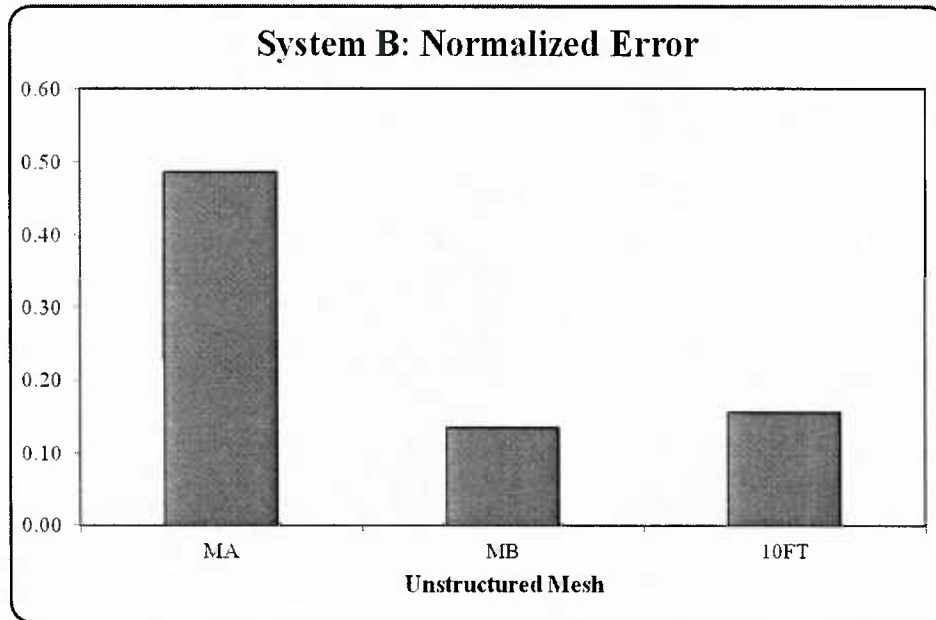


Figure 42. Error single CPU/GPU full solution modeled performance (**System B**).

Table 17

*Performance modeling single CPU/GPU full solution (**System A**)*

Unstructured Mesh	Actual Time (secs)	Estimated Time (secs)
MA	204.69	157.30
MB	1,804.05	1,760.76
10FT	140.55	135.78

Table 18

*Performance modeling single CPU/GPU full solution (**System B**)*

Unstructured Mesh	Actual Time (secs)	Estimated Time (secs)
MA	146.64	75.25
MB	974.90	843.17
10FT	89.23	75.29

4.6.3 Contribution of Hardware Factors. This section establishes a relationship to hardware factors and the resulting application performance via the derived equation (4.9.3), adjusting hardware variables and then projecting against the actual performance of the

application. The resulting differentials are analyzed and the impact of the adjusted parameter(s) on performance of the CPU/GPU computing system is theorized.

4.6.3.1 Single PCG Call. The first performance modeling that is adjusted for hardware factor(s) changes is the single call to the PCG solver, since this is used to build the final full-solution model (see equations (4.8) and (4.9.3)). The number of SMP chips was adjusted to be greater than and less than the current number defined in both single CPU/GPU computing systems for a single call to the PCG iterative solver to determine the theoretical effect the SMP counts have on the resulting performance.

The behavior expressed by both **System A** and **System B** computing systems was similar, excepting of course the *faster* nature of **System B** [58-61, 76]. Figure 43 shows that reducing the number of SMPs to 4 for **System A** resulted in an increase in execution time, exacerbated by the larger input of unstructured mesh **MB**. This same reduction in SMPs on **System B** shown by Figure 44 resulted in a similar increased execution time and, as with **System A**; the larger model configuration **MB** had a greater impact. The observed performance change for both **System A** and **System B** as a result of this adjusted hardware factor is as expected given the larger model mesh **MB** presents more elements for a relatively low number of SMPs – SMPs are the physical equivalent of the logical block [37, 62] so lower counts manifest less computational power that will be further debilitated applied in larger problem space provided by input mesh **MB**. The converse is observed by increasing the number of SMPs to 64 for both **System A** and **System B**, as before this is no source of consternation – more computational power will naturally be leveraged by a larger environment in which to be expressed.

The hardware factor of SMPs at the level of a single call to the PCG iterative solver extends naturally to the performance of the full candidate solution. The full candidate solution subsumes the single PCG call and works in tandem - result expressed as a magnification, a constant, or abrogation to resulting performance. The full candidate solution hardware factors for the single CPU/GPU computing system are discussed in the next sub-section.

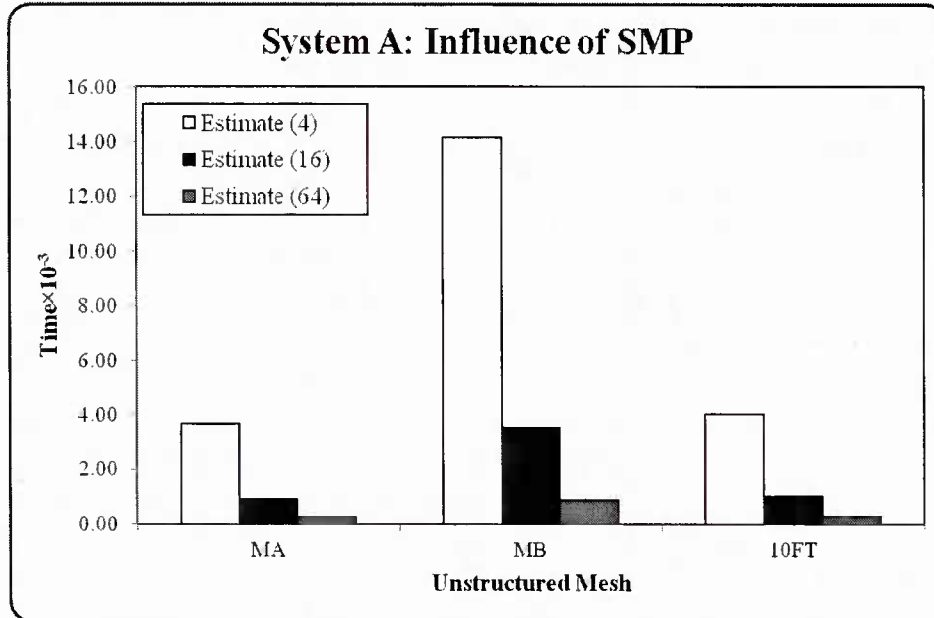


Figure 43. Performance model for single PCG solver on **System A** (SMP adjusted).

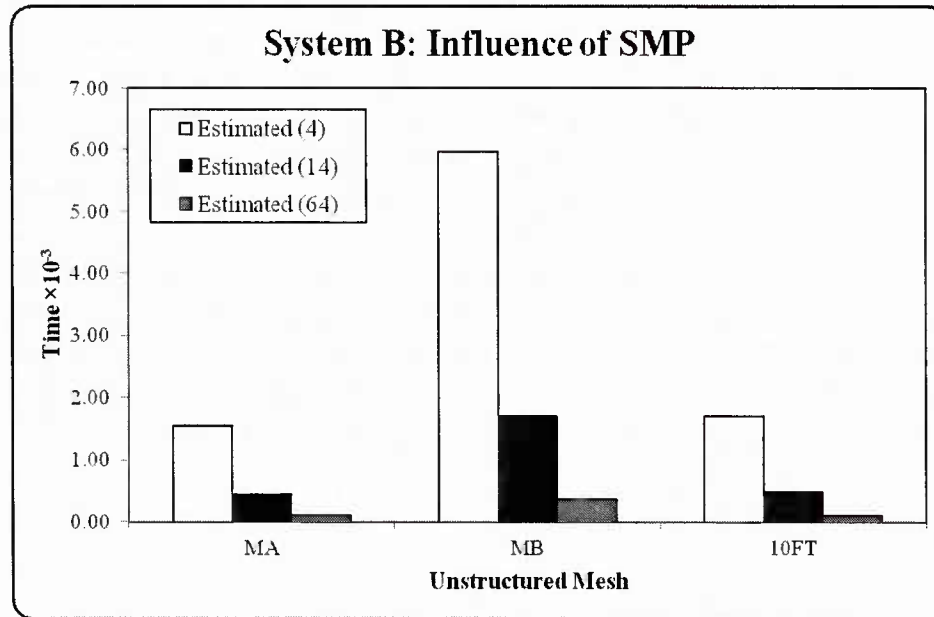


Figure 44. Performance model for single PCG solver on **System B** (SMP adjusted).

4.6.3.2 Full solution – single CPU/GPU. Adjusting the hardware factor(s) for the full candidate solution and the resulting impact(s) on the performance is modeled in this sub-section. As with the previous sub-section the number of SMPs is adjusted and the resulting theoretical performance is examined to deduct the relative impact of this factor on behavior for both single

CPU/GPU computing environments for **System A** and **System B**. Equation (4.9.2) defines the most direct access of the number of SMPs to the full candidate solution as a rate of growth/decay.

The full solution performance model given by (4.9.3) applies the Gaussian distribution to the overall performance of the system with the determination of serialized threads within a warp defined as the allocated register unit size for the given GPU device - the ratio of SMPs directly impact the value of this allocated unit size as fewer SMPs coerces lower unit sizes and less computational ability and vice versa. SMPs are given this position in the performance complexity model as each will provide a factor to the initial single PCG solver call, modeled by equation (4.8). Examining the performance results in Figure 45 and Figure 46 reveal a comparable behavior to that observed for the single PCG call discussed previously but to a much less magnitude.

The single CPU/GPU computing system environments defined by **System A** and **System B** express a performance boost for an increased number of SMPs and a higher execution time for decreased numbers of SMPs. However, the full candidate solution is less affected by this hardware factor change as was with the single PCG call with input mesh configuration **MA** yielding negligible results for **System A** and **System B**; the input mesh **10FT** mesh configuration now reflects the performance comparable to mesh configuration **MB**.

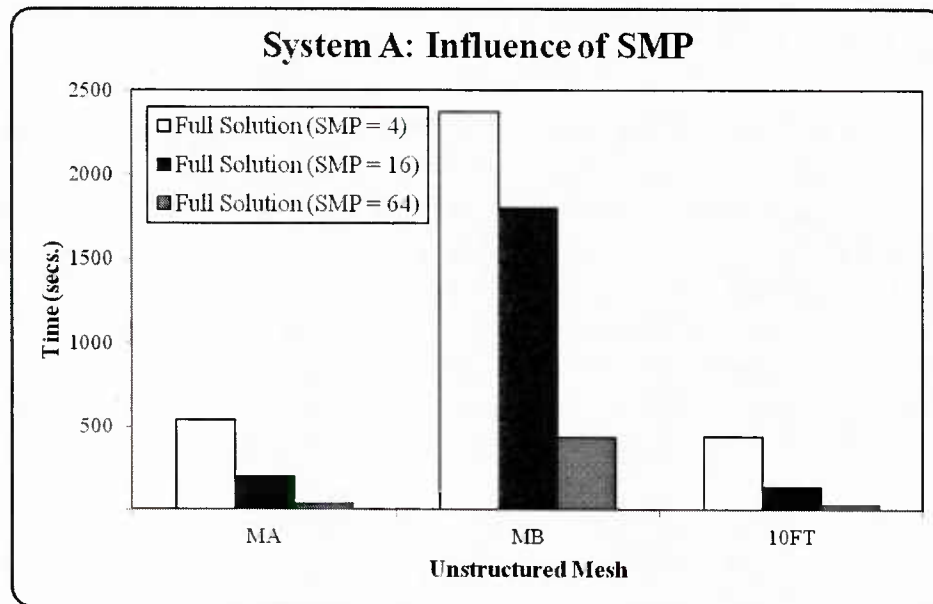


Figure 45. Performance model for full candidate solution on System A (SMP adjusted).

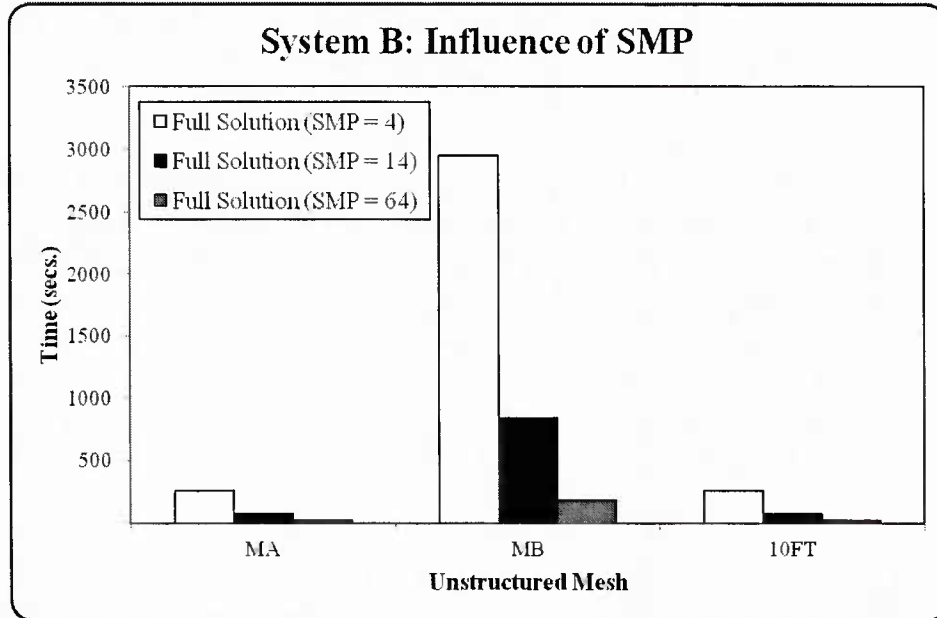


Figure 46. Performance model for full candidate solution on **System B** (SMP adjusted).

4.6.4 Contribution of Software Factors. This section establishes a relationship to software factors and the resulting application performance via the derived equation (4.9.3), adjusting hardware variables and then projecting against the actual performance of the application. The resulting differentials are analyzed and the impact of the adjusted parameter(s) on performance of the CPU/GPU computing system is theorized.

4.6.4.1 Single PCG Call. The number of threads per block is an obvious software factor to adjust as this can be related to memory address coalescing and can be viewed as another, less direct application of thread occupancy. The theoretical performance results of both **System A** and **System B** corroborates documented performance regarding number of threads per block [48, 49, 65, 66]. Lowering the number of threads per block to 128 from the optimal 256 increases theoretical execution time for the single PCG solver and increasing to 768 boosts performance, modeling the effect of hardware coalescing of memory addresses [62, 77]. This software factor is manifest for both **System A** and **System B** computing environments as shown in Figure 47 and Figure 48 for **System A** and **System B** respectively.

The influence of the given software factors is clearly evident at the single PCG solver call level, the next sub-section discusses the full candidate solution.

4.6.4.2 Full solution – single CPU/GPU. Adjusting software factor(s) for the full candidate solution and the resulting impact(s) on performance is modeled in this sub-section – the number of threads per block is altered. The effect of this software factor is similar to the

observed results for the single PCG solver call - smaller numbers of threads express lower performance whereas greater numbers yield a performance boost.

The observed hardware and software factors work together with the defined computational algorithm to effect performance – this interplay of factors is discussed in the next section.

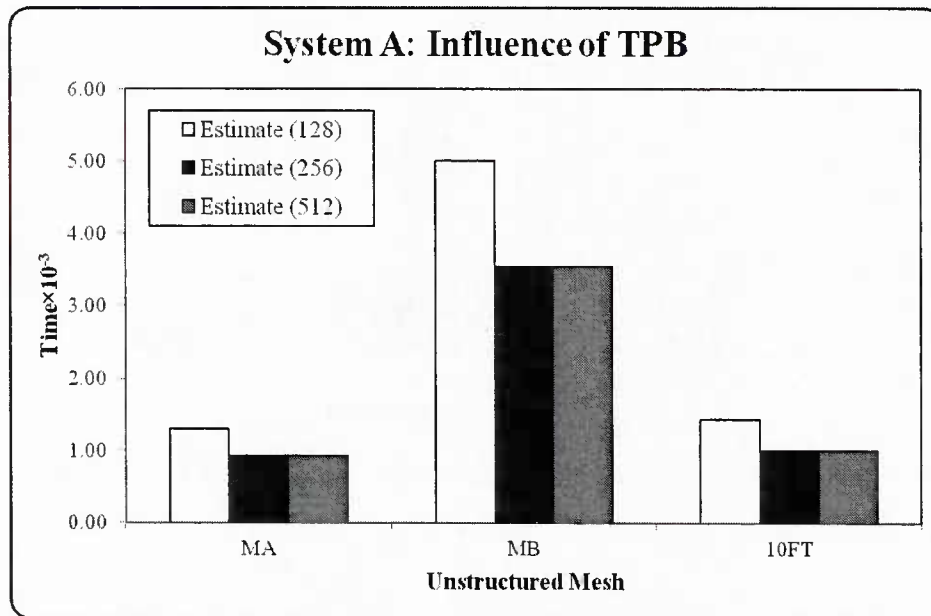


Figure 47. Performance model for single PCG solver on **System A** (Threads adjusted).

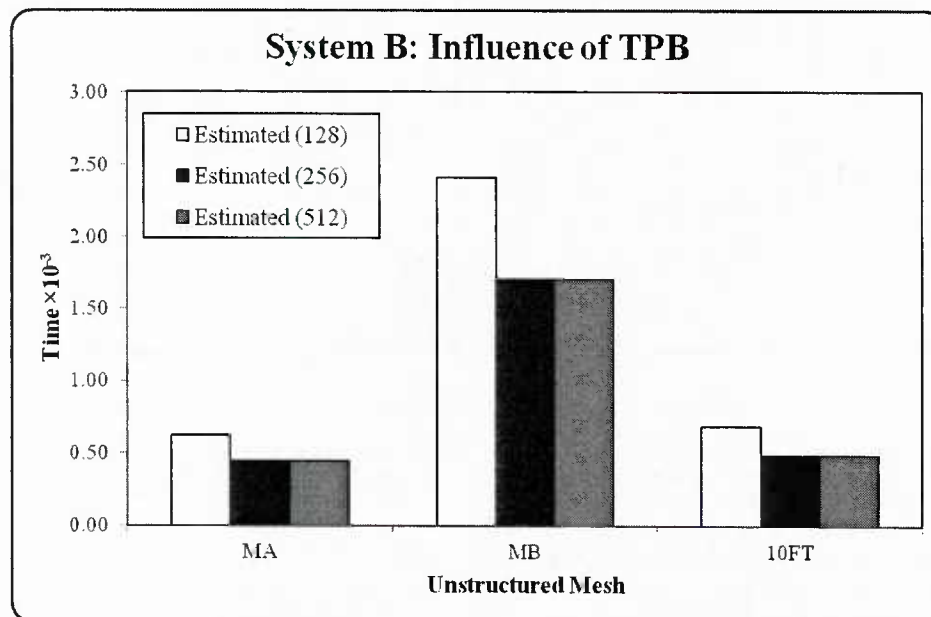


Figure 48. Performance model for single PCG solver on **System B** (Threads adjusted).

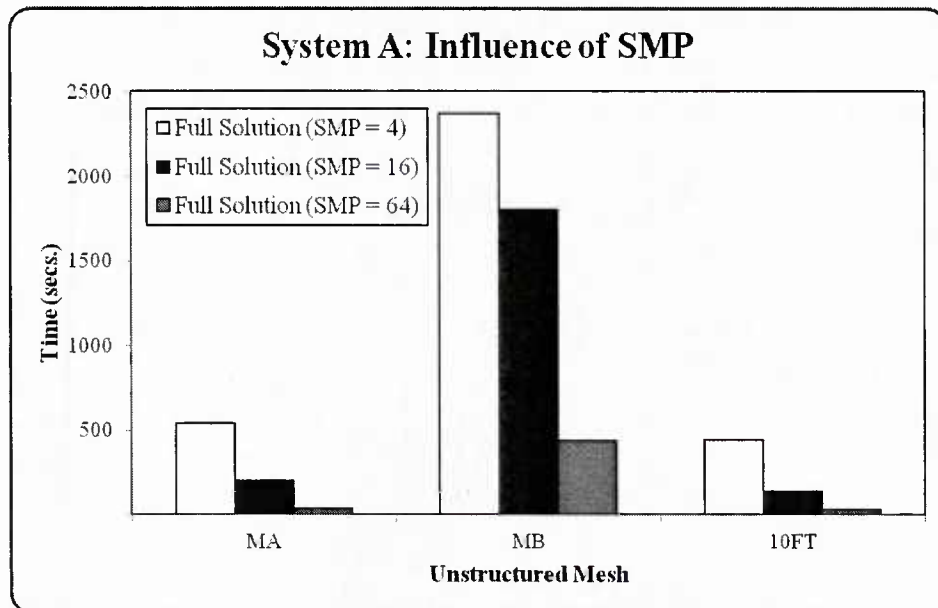


Figure 49. Performance model for full candidate solution on **System A** (SMP adjusted).

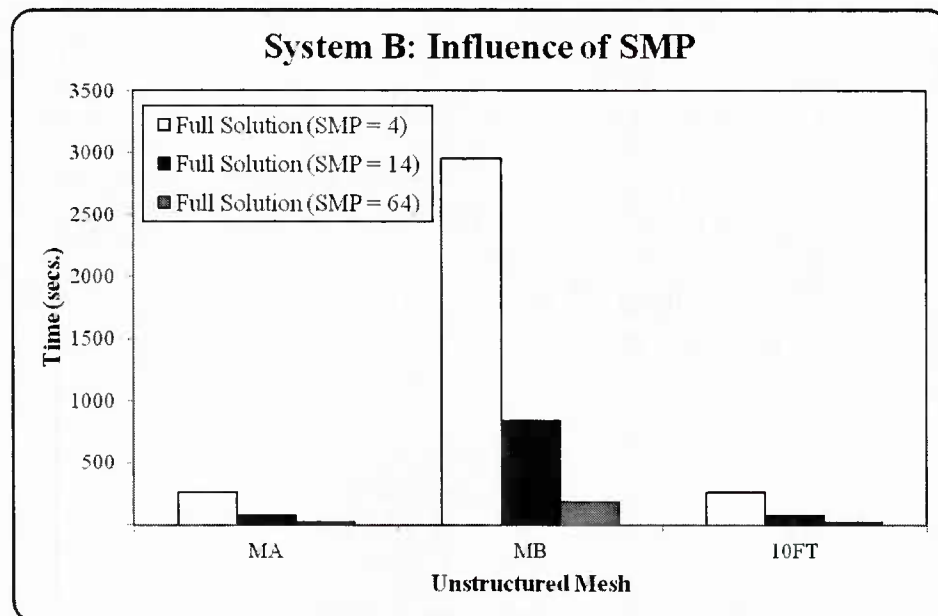


Figure 50. Performance model for full candidate solution on **System B** (SMP adjusted).

4.7 Performance and Relation to Software and Hardware Factors

The resulting performance of the *single* CPU/GPU computing system is directly tied to the interplay of software and hardware factors of the environments in which they executed and are related in this sub-section. Hardware factors such as the addition of SMPs and increasing

memory provide the context in which software factors can express further optimization and software factors such as increasing thread occupancy and data compression formats that can increase locality allow device architectures to apply low-level features like hardware multi-threading for maximal memory throughput.

Equation (4.9.3) accommodates a mathematical model to map theoretical performance changes as *hardware* and *software* variables are altered, allowing the developer to better understand the effects different software and hardware constructs for HPC computational modeling applications using CPU/GPU computing systems. The development of predictive models for computing follows three general approaches – analytical, profile, and simulation based categories [92].

Simulation-based performance prediction uses an application that has modeled the objective architecture in exacting detail and generates results from dynamic and random inputs – accurate but computationally costly [92]. Profile-based performance prediction uses two stages to develop the model; instrumentation is utilized to generate statistical information on a given program run and analysis is used on these statistics to create an estimation of performance for a given architecture [92]. The analysis-based performance prediction model derives a mathematical equation that can estimate program behavior for specific architectures and algorithms – this is the model utilized in this chapter and follows precepts established by the works of [66, 72, 92, 93].

All of the following discussions on the effects of *hardware* and *software* factors and any resulting interplay are derived from altering the variables of equation (4.9.3) and illustrate theoretical performance as consequence. The software factor **Threads-Per-Block** are denoted as **TPB** and the theoretical change exhorted by different parameters is given as **ETime** in the following discussion.

Table 20 categorizes the hardware factors with **System A** as the computing environment contrasting the actual solution execution time against the effective time generated by the alteration of input variables and is shown in milliseconds. *Table 21* categorizes the software factors with **System A** as the computing environment contrasting the actual solution execution time against the effective time generated by the alteration of input variables and is shown in milliseconds.

Table 22 categorizes the hardware factors with **System B** as the computing environment contrasting the actual solution execution time against the effective time generated by the alteration of input variables and is shown in milliseconds. *Table 23* categorizes the software factors with **System B** as the computing environment contrasting the actual solution execution

time against the effective time generated by the alteration of input variables and is shown in milliseconds.

The degree of impact that the proper implementation of software and hardware factors present is shown in *Table 18* and *Table 19* for **System A** and **System B** respectively. The resulting factors of change are calculated as *non-dimensional quantifier* such that differences from the original solution times are applied against the solution times that result when the parameter/factors are expressed – the closer to zero the less effect the factor has on system performance and the greater the magnitude beyond one the more negative the effect on performance. The larger negative impact on performance is manifest as the lowering of the number of SMP, the converse is also true and is expected given that the increasing of SMP invariably increases the number of graphics cores from which to utilize in a given problem domain.

Interestingly the relative effect of increasing the number of TPB has the same consequence regardless of the computing system utilized. The relative equivalence of software factors regardless of computing environments is to be expected as the algorithm should operate independently if optimized – of course, this is not conclusive as the more dispersed the non-zero elements, the greater the irregularity of memory addressing, memory bank conflicts and the resultant performance degradation. This is evidenced by *Table 18* as the lowered number of SMPs exposes the weakness of locality – higher relative latency due to lower hardware ability to hide it. Regularity of the sparse structure can play a significant impact to the potential performance benefit in CPU/GPU computing environments.

However, this is not conclusive as the analytical performance model derived is, as all prediction models of this category, based on *conservative measures* of a given architecture and problem domain - extrapolating too far beyond the initial derivation can create computational artifacts in the results.

Table 19

Categorized software and hardware effects (System A)

Input	Hardware Factor	Value	Factor of Change	Software Factor	Value	Factor of Change
MA	SMP	4	2.427	TPB	128	0.414
MB	SMP	4	0.352	TPB	128	0.414
10FT	SMP	4	2.272	TPB	128	0.414
MA	SMP	64	0.750	TPB	512	0.000

MB	SMP	64	0.750	TPB	512	0.000
10FT	SMP	64	0.750	TPB	512	0.000

Table 20

Categorized software and hardware effects (System B)

Input	Hardware Factor	Value	Factor of Change	Software Factor	Value	Factor of Change
MA	SMP	4	2.50	TPB	128	0.414
MB	SMP	4	2.50	TPB	128	0.414
10FT	SMP	4	2.50	TPB	128	0.414
MA	SMP	64	0.780	TPB	512	0.000
MB	SMP	64	0.780	TPB	512	0.000
10FT	SMP	64	0.780	TPB	512	0.000

Table 21

Hardware factors and theoretical performance (System A)

Input	Variable	Value	Time (ms.)	ETime (ms.)
MA	SMP	4	204,689.00	537,490.96
MB	SMP	4	1,804,050.00	2,370,321.50
10FT	SMP	4	140,546.00	444,203.64
MA	SMP	64	204,689.00	39,324.91
MB	SMP	64	1,804,050.00	440,189.68
10FT	SMP	64	140,546.00	33,945.25

Table 22

Software factors and theoretical performance (System A)

Input	Variable	Value	Time (ms.)	ETime (ms.)
MA	TPB	128	204,689.00	222,455.30
MB	TPB	128	1,804,050.00	2,490,088.86
10FT	TPB	128	140,546.00	192,023.31
MA	TPB	512	204,689.00	204,689.00
MB	TPB	512	1,804,050.00	1,804,050.00
10FT	TPB	512	140,546.00	140,546.00

Table 23

Hardware factors and theoretical performance (System B)

Input	Variable	Value	Time (ms.)	ETime (ms.)
MA	SMP	4	146,635.00	263,356.17
MB	SMP	4	974,897.00	2,951,083.33
10FT	SMP	4	89,228.10	263,514.70
MA	SMP	64	146,635.00	16,459.76
MB	SMP	64	974,897.00	184,442.71
10FT	SMP	64	89,228.10	16,469.67

Table 24

Software factors and theoretical performance (System B)

Input	Variable	Value	Time (ms.)	ETime (ms.)
MA	TPB	128	146,635.00	106,411.96
MB	TPB	128	974,897.00	1192417.74
10FT	TPB	128	89,228.10	106,476.02
MA	TPB	512	146,635.00	146,635.00
MB	TPB	512	974,897.00	974,897.00
10FT	TPB	512	89,228.10	89,228.10

The performance results of the single CPU/GPU computing systems have been shown to be consistent for presented input unstructured mesh model configurations with varying sizes for both **System A** and **System B** computing systems. This performance can be dramatically affected by sometimes slight aberrations of input – from this *single* CPU/GPU paradigm the *multiple* CPU/GPU methodology is analyzed and discussed in the next chapter.

CHAPTER 5

Full Candidate Application – Multiple CPU/GPU Computing System

This chapter focuses on the full solution to the candidate application within the context of a *multiple* CPU/GPU computing system for both **System A** and **System B**. The full solution of the computationally intensive candidate application is mapped to the CPU/GPU computing system, distributed across independent nodes, and the resulting performance is analyzed to determine how the hardware and software factors work together to impact the resulting application performance. During the mapping, key computationally intensive kernels are presented and associated GPU developments explored.

This chapter will ascertain how the hardware architectures of **System A** and **System B** work together with the software factors to denote the application performance – key in this discussion is the calculation of a computational complexity analysis for *multiple* CPU/GPU computing systems which is a natural extension from the single version presented in the previous chapter. The computational complexity analysis is actualized as a performance modeling equation that can be used to project how different problem, software, and hardware parameters will affect performance. Specific computational behavior of multiple CPU/GPU computing systems is the exposure of the important cost of intra-nodal and local host communication to the performance of a computationally intensive application.

Understanding these variations in factors/parameters is essential as new computing architectures arrive to get optimal performance for HPC computational modeling legacy and new code development applications.

5.1 Mapping Full Candidate Application to GPU

The mapping of the full candidate application to the *multiple* CPU/GPU environment extends from the previous chapter, detailing the mapping of the *single* CPU/GPU computing system, and is developed and demonstrated for both **System A** and **System B** environments. The previous chapter applied the CNC solver set in the context of a shared memory address environment and this chapter grows the environment to include *multiple* systems each with its own CPU/GPU architecture providing intra-node communication via MPI standard [28, 94]. The ability of CPU/GPU computing systems to scale with *multiple* architectures is critical as HPC applications have long embraced the increased performance provided by parallelizing large-scale

problems with domain decomposition techniques via MPI and much research in the GPGPU computing community is targeting this objective [36, 69, 95-97]. The *single* CPU/GPU computing system presented in the previous chapter passed the computationally intensive solution to system of linear equations in matrix form $\underline{Ax} = \underline{b}$ to a local GPU device where it can be most beneficial - extending this paradigm to the *multiple* systems encompasses an extra level of intra-nodal communication, e.g. MPI.

MPI is a standard for message passing systems with different implementations such as MVPICH [26, 28] and historically dominates HPC modeling as an effective methodology for application performance boosting [26, 28]. Implementations of the MPI standard define a tool for connecting multiple machines and/or discrete CPUs as a logical whole in order to solve problems that are computationally prohibitive in a single machine context [26, 28]. This methodology is similar to the **Parallel Virtual Machine** (PVM), the predecessor of MPI [26, 28], but the differences are important.

PVM and MPI take different approaches as to the defining and utilization of distributed topologies. MPI allows the user to easily create *virtual* topologies [98] that must be explicitly set in PVM [99, 100]. The abstraction of topologies with MPI is one of the reasons for its popularity; software developers do not have to focus on different architectural environments when creating an application in MPI. The use of virtual topologies has another benefit – many MPI implementations optimize the definition of the system based on the physical nodes in the current cluster. The MPI implementation simply alters the identifications of the various processors contained in the defined communicator to reflect the optimal distances of the actual machines contained in the system [98, 100]. PVM will allow for the communication, not only between heterogeneous architectures but also between different languages – e.g., a C-Code program can interface to a FORTRAN-Code program and vice-versa. PVM will probe for differences in architecture to allocate native resources as needed. MPI, whose chief design is around both performance *and* portability, assumes a consistent connection via a defined world communicator [98, 100]. PVM is designed for operation within a heterogeneous set of architectures, while MPI *can* do this also, it is not explicitly defined within the standard itself [98, 100].

MPI was chosen as the standard for intra-node communication for the presented candidate composite process flow modeling application as this represents a significant body of research in GPGPU computing [101-104] and porting legacy code to utilize CPU/GPU systems requires a robust and portable solution that encompasses this paradigm. The key computationally intensive Kernels encompassed by the *multiple* CPU/GPU computing systems are discussed next.

5.1.1 Key Computationally Intensive Kernels. The key computationally intensive Kernels for the *multiple* CPU/GPU computing systems introduced in this chapter are evolved from two distinct sources. The first source is a direct result of the utilization of sets of single CPU/GPU computing systems from which the multiple CPU/GPU computing system is formed as each individual CPU/GPU machine involved brings with it the local computationally intensive Kernels that are effected by a separate machine architecture. The second source is directly related to the communication vehicle for the *multiple* CPU/GPU computing systems, MPI and any communicational overhead it brings to the total system is magnified by the significant GPU and CPU communications via the local PCIe bus – a noted bottleneck in single CPU/GPU systems now magnified by the number of distinct nodes in the communication world of MPI [69, 97].

5.1.2 GPU Code Developments. This sub-section establishes GPU code developments such as API tools/libraries and the data-structures/layouts for the definition of the multiple CPU/GPU computing system. Nvidia’s CUDA API has remained ahead in the GPGPU computing community [1, 2, 24] and this continues as interest grows in CPU/GPU computing clusters with the provision of the **Unified Virtual Address (UVA)** space [37, 62, 105].

UVA allows CUDA to map GPU device buffers into a global virtual address space and then queries the system to determine if a desired address is in GPU or CPU space. UVA then signals CUDA to execute a PCIe communication or local memory call, depending on the physical location of the virtual address – theoretically allowing for direct MPI buffer transfers. However, CUDA UVA was not utilized with the presented candidate application as the construct did not come to fruition until compute architecture 2.0 and this leaves out **System A** [58]. So in the interest of consistency was not pursued in this dissertation.

Given that the individual GPU devices in the multiple CPU/GPU computing systems have no *direct* communication to outside nodes a *double-copying* is inferred [105]. The individual CPU/GPU computing system executed upon a *local* sub-domain of the *global* problem space, as per domain decomposition [26, 28], passing the computationally intensive system of linear equations defined as a sparse matrix system to the local GPU. The GPU executes the system using the PCG iterative solver, as with the single CPU/GPU system, and returns the result back across the PCIe bus to the CPU where it is then stored in the defined MPI communication buffers to be shared with other nodes in the system – unavoidably increasing the latency as there must now be explicit staging of memory buffers for collective and point-to-point calls [94] and GPU to CPU to MPI and back the same path at each iteration of the algorithm.

The full candidate application is validated against an analytically derived solution for a simple 2D radial injection circular plate mold geometry model using the *multiple* CPU/GPU computing system next.

5.2 Validation of Full Candidate Application on Multiple CPU/GPU

The correctness of the full candidate application for the *multiple* CPU/GPU computing systems is ensured via the examination of flow-front progression and injection port pressures of numerical solutions for CPU and GPU against the correspondingly analytical results. As before, the model used for validation is a simple 2D circular plate with the resulting analytical equation.

The simple model being used in this chapter is a radial flow in a circular plate with a radius of 10 cm and an inner radius of 0.15 cm shown in Figure 56. The inner radius, R_0 , is subjected to a constant flow rate Q . The thickness of the cavity is H , the pressure is P , resin viscosity is μ , the permeability of the fiber preform is \bar{K} , and the porosity of fiber compaction is ϕ . The flow front radius at any time t is given by [89]:

$$R(t) = \left(\frac{Qt}{\pi\phi H} + R_0^2 \right)^{\frac{1}{2}} \quad (5.1)$$

The corresponding expression for injection pressure, which varies with time, is given by [89]:

$$P_0 = \left(\frac{\mu Q}{2\pi \bar{K} H} \ln \left(\frac{R(t)}{R_0} \right) \right) \quad (5.2)$$

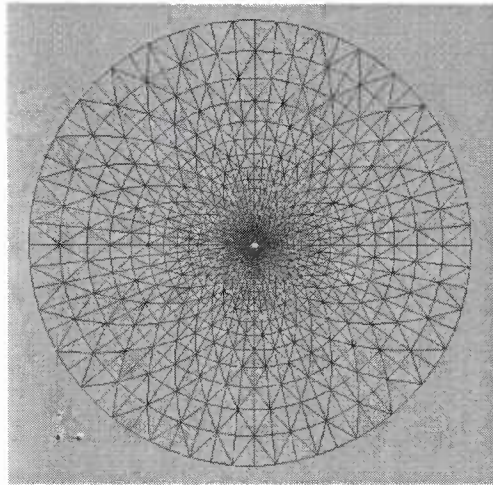


Figure 51. 2D circular plate validation model (not to scale).

The following physical parameters are used in this analysis:

$Q = 2.4 \frac{\text{cm}^3}{\text{sec}}$, permeability $\bar{K} = 44.0e-08 \text{cm}^2$, a viscosity $\mu = 0.02 \text{PaS}$, a porosity of $\phi = 0.805$, a time step $\Delta t = 0.5 \text{sec}$, and an element thickness $H = 0.742 \text{cm}$. The circular plate model involved a computational mesh of 1,344 nodes and 2,560 3-noded triangular elements. Figure 52

and Figure 54 display the flow-front progression for **System A** and **System B** respectively and clearly define accuracy with the analytical value. Figure 53 and Figure 55 display the inlet injection pressure for **System A** and **System B** respectively and clearly define accuracy with the analytical value.

The flow-front and inlet injection pressure values are accurate so the full candidate solution for the multiple CPU/GPU computing systems is presented next with a focus on initial performance evaluation using the *multiple* CPU/GPU computing systems.

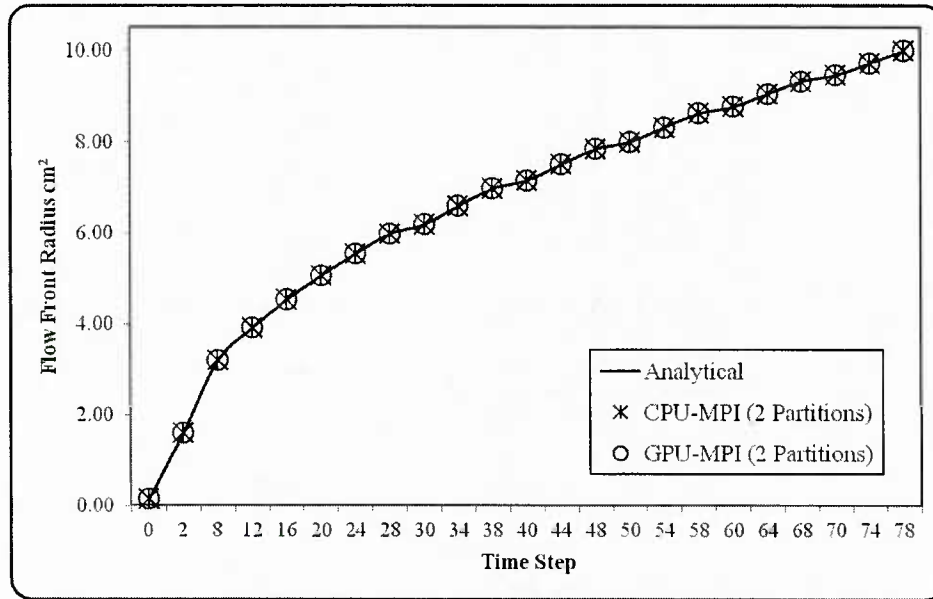


Figure 52. Validation of multiple CPU/GPU for flow-front progression (**System A**).

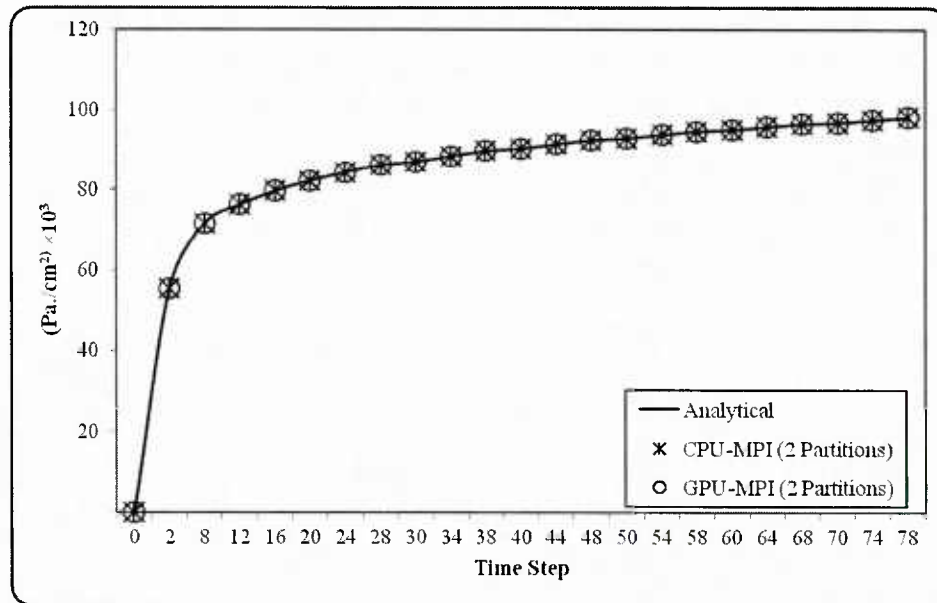


Figure 53. Validation of multiple CPU/GPU for inlet injection pressure (**System A**).

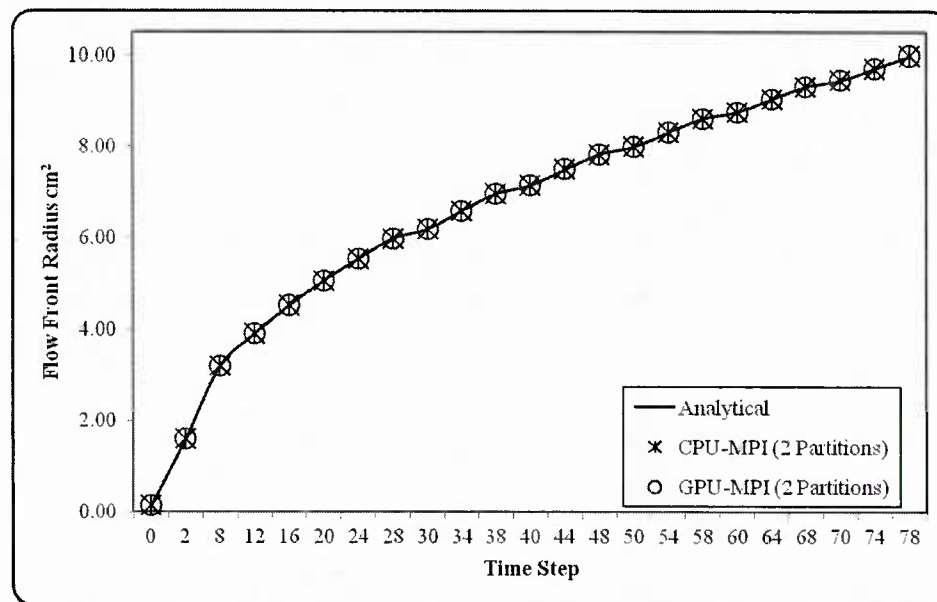


Figure 54. Validation of multiple CPU/GPU for flow-front progression (**System B**).

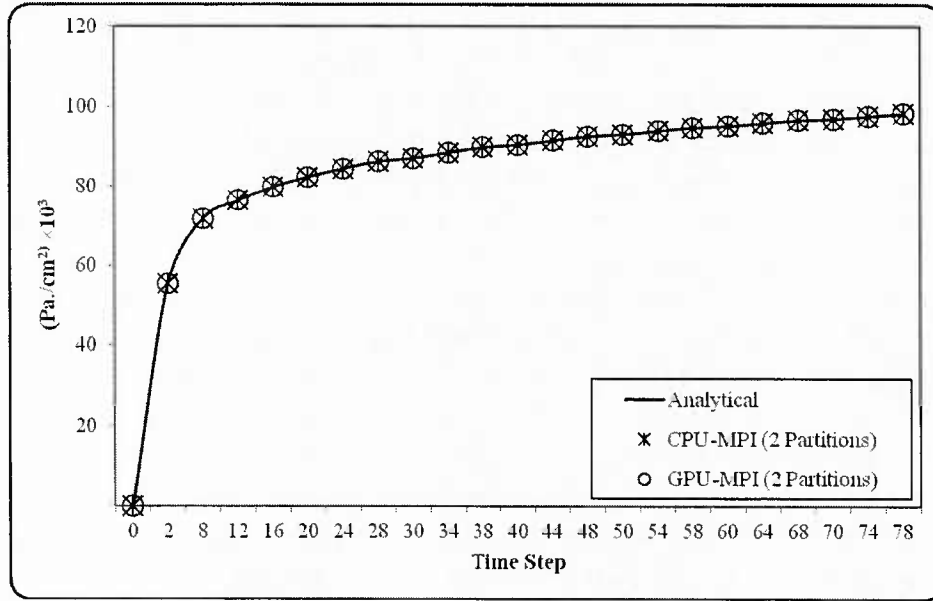


Figure 55. Validation of multiple CPU/GPU for inlet injection pressure (System B).

5.3 Initial Full Candidate Application Performance on Multiple CPU/GPU

Much of the establishment of key computationally intensive Kernels for the multiple CPU/GPU computing systems is a reflection of the *single* CPU/GPU computing system from the previous chapter. The single system provides the building blocks of the multiple systems leaving the effective performance of the *multiple* systems to the parlance of latency mitigation as a product of standard MPI communication overhead [26, 28] and noted CPU/GPU costs [106-109].

5.3.1 System A. The full candidate solution for *multiple* CPU/GPU computing systems was applied using **System A** and is discussed in this sub-section. The initial problem domain was partitioned into various sub-domains each to be executed on a single CPU/GPU computing system. The multiple sub-domain results are then compared against the CPU-only, and the CPU/MPI solution times.

Table 24 shows the total solution times, in milliseconds, for increasing numbers of partitions using as input unstructured mesh **MA** expressed using CSR data format compression with **System A** computing architecture. The observed total solution times in all cases are higher for the multiple CPU/GPU computing system over the single CPU-only and CPU plus MPI which illustrates the significant cost potential of the extra layer of latency produced by intra-node communication within the execution of the global solution domain. The slight decrease at 2-partitions for the GPU plus MPI model is due to the cache effect of increased local memory to allow more of the problem to be immediately available for solving.

Table 25 shows the total solution times, in milliseconds, for increasing numbers of partitions using as input unstructured mesh configuration **MB** expressed using CSR data format compression with **System A** computing architecture. The observed total solution times for GPU plus MPI are more promising as the increased computational loads on the individual nodes using the larger sized input mesh are large enough to overcome the latency of intra-node as well as the PCIe bottleneck – until 16 sub-domains are created and computational loads on individual nodes in the system become too poor to overcome increasing latency for this fixed problem size. Figure 56 and Figure 57 visually depict the observed performance using **System A** as the computing environment and different sized mesh configurations **MA** and **MB** respectively.

The more structured meshes **MA** and **MB** depict distinct performance differences when employing MPI with the local GPU. The smaller model **MA** illustrates no performance benefit for MPI with GPU over MPI without GPU as the smaller model does not have enough computational intensity at the local GPU level to overcome the cost of intra-nodal communication generated by MPI [70, 105]. The larger model **MB** reveals a slight performance boost when using MPI and the local GPU for the global count of partitions that remain below 16 when the intra-nodal communication cost once again become the dominating factor of performance. The more unstructured mesh model **10FT** shows a negligible difference of MPI with or without utilizing the local GPU due to the more evenly distributed non-zero elements, each computing node in the system is given nearly equal divisions of work and the so throughput latency is better handled, thus hiding the intra-nodal communication cost of MPI better.

Table 26 shows the total solution times, in milliseconds, for increasing numbers of partitions using as input unstructured mesh configuration **10FT** defined earlier expressed using CSR data format compression with **System A** computing architecture. The observed total solution times for GPU plus MPI are higher than the corresponding CPU plus MPI – the performance is closer than the results shown by the mesh **MA** as the numbers of non-zero elements is greater for **10FT** but the intra-nodal communication derived from the coarser-grained parallelism of MPI communication creates higher levels of latency that must be mitigated [69, 105, 109].

The next sub-section will discuss the results of the multiple CPU/GPU architecture defined by **System B**.

Table 25

Multiple CPU/GPU performance in milliseconds with mesh MA (System A)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	2,749,450.00	418,440.00
CSR	2	1,666,960.00	1,604,200.00
CSR	4	825,989.00	1,357,890.00
CSR	16	260,669.00	1,461,590.00

Table 26

Multiple CPU/GPU performance in milliseconds with mesh MB (System A)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	77,163,900.00	4,219,610.00
CSR	2	20,169,400.00	16,985,600.00
CSR	4	9,633,960.00	8,570,300.00
CSR	16	2,495,580.00	23,956,100.00

Table 27

Multiple CPU/GPU performance in milliseconds with mesh 10FT (System A)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	6,649,810.00	140,546.00
CSR	2	1,206,160.00	1,246,120.00
CSR	4	621,867.00	688,750.00
CSR	8	321,710.00	545,930.00

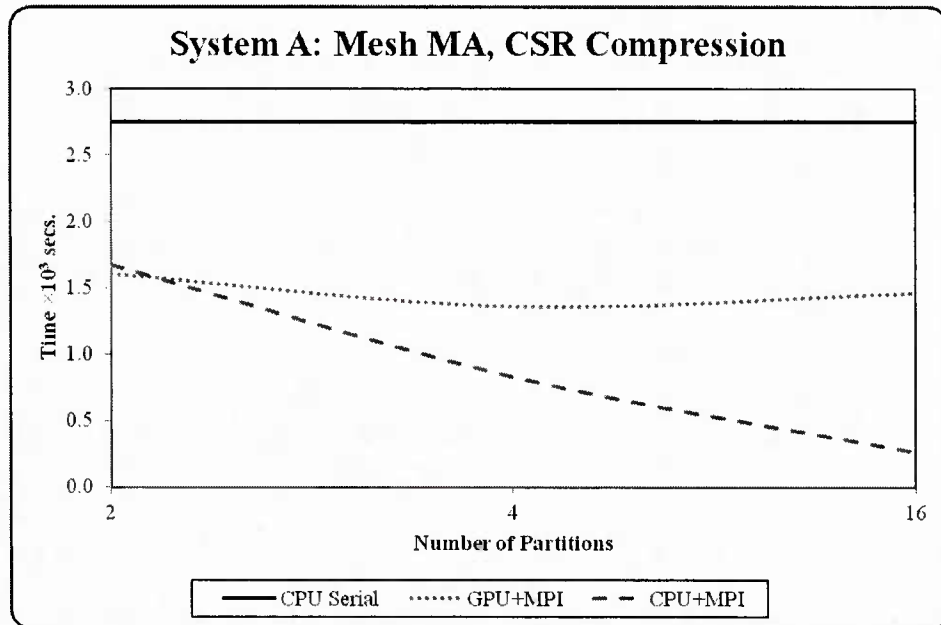


Figure 56. Multiple CPU/GPU computing system - mesh MA (System A).

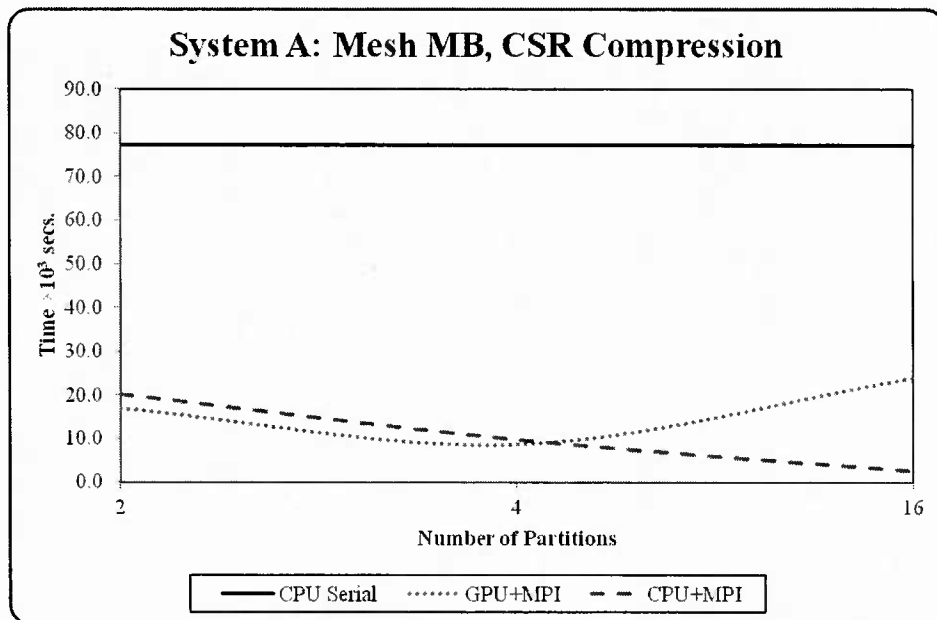


Figure 57. Multiple CPU/GPU computing system - mesh MB (System A).

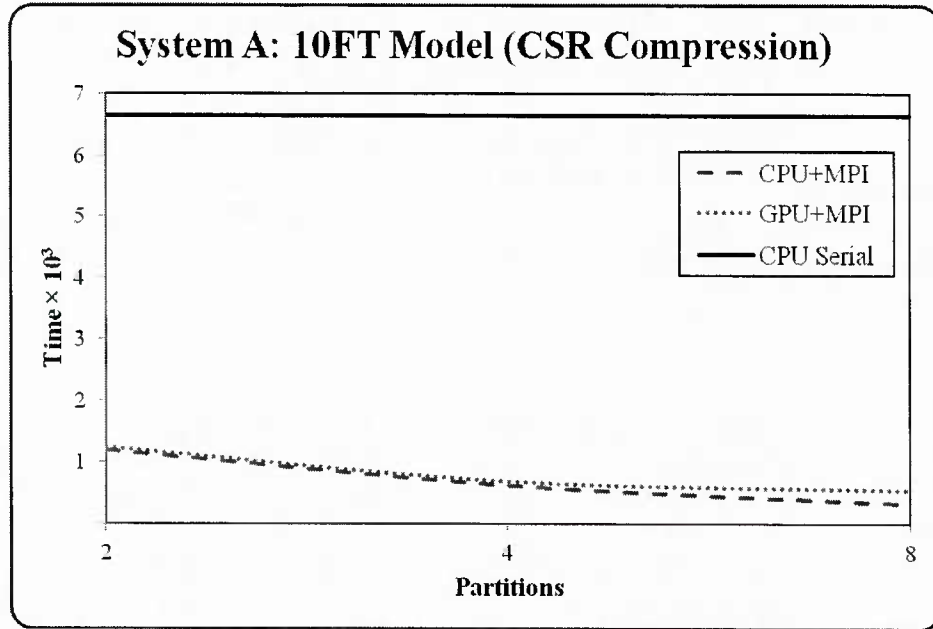


Figure 58. Multiple CPU/GPU computing system - mesh 10FT (System A).

5.3.2 System B. The full candidate solution for multiple CPU/GPU computing systems is executed in the **System B** computing environment and is discussed in this sub-section. The initial global problem domain is partitioned into various sub-domains and passed among various discrete processing nodes to be executed in the manner of a single CPU/GPU computing system. The multiple sub-domain results are then compared against the CPU-only, and the CPU/MPI solution times.

Table 27 shows the total solution times, in milliseconds, for increasing numbers of partitions using as input unstructured mesh configuration **MA** expressed using CSR data format compression with **System B** computing architecture. The observed total solution times in all cases involving sub-domain partitions reveal that the GPU plus MPI construct outperforms the CPU/MPI model. However this positive benefit of combining GPU and MPI is not observed in the larger input mesh configuration **MB** as can be seen in *Table 28*. This observed performance degradation for the larger element mesh is a stark contrast from the behavior manifested in **System A**, where a larger mesh resulted in better performance as the computationally intensive operations increased.

Table 29 shows the total solution times, in milliseconds, for increasing numbers of partitions using as input unstructured mesh configuration **10FT** expressed using CSR data format compression with **System B** computing architecture. The observed solution times for the **10FT** model, while executing on the more advanced CPU/GPU computing **System B** never manages to overcome the latency incurred by the intra-nodal communication emergent from the use of

multiple MPI communication calls – evidence of a faster and more efficient hardware that minimizes costs at the local host and conversely *exposing* the MPI communication costs.

Figure 59 and Figure 60 are visual depictions of the observed results of the multiple CPU/GPU architecture **System B** listed in *Table 27* and *Table 28*, and Figure 61 illustrates the data given in *Table 29*. The next sub-section will examine, analyze and discuss the observed initial performance results for multiple CPU/GPU systems represented by computing **System B** and **System A**.

Table 28

Multiple CPU/GPU performance in milliseconds with mesh MA (System B)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	420,980.00	168,570.00
CSR	2	767,364.00	220,462.00
CSR	4	448,665.00	98,034.20
CSR	16	307,081.00	44,392.40

Table 29

Multiple CPU/GPU performance in milliseconds with mesh MB (System B)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	6,306,240.00	1,197,350.00
CSR	2	1,675,850.00	2,995,870.00
CSR	4	872,019.00	2,172,520.00
CSR	16	318,895.00	1,490,310.00

Table 30

Multiple CPU/GPU performance in milliseconds with mesh 10FT (System B)

Data Compression	Partitions	CPU + MPI (ms.)	GPU + MPI (ms.)
CSR	1	616,770.00	89,228.10
CSR	2	124,995.00	249,532.00
CSR	4	73,081.40	145,577.00
CSR	8	41,426.60	118,746.00

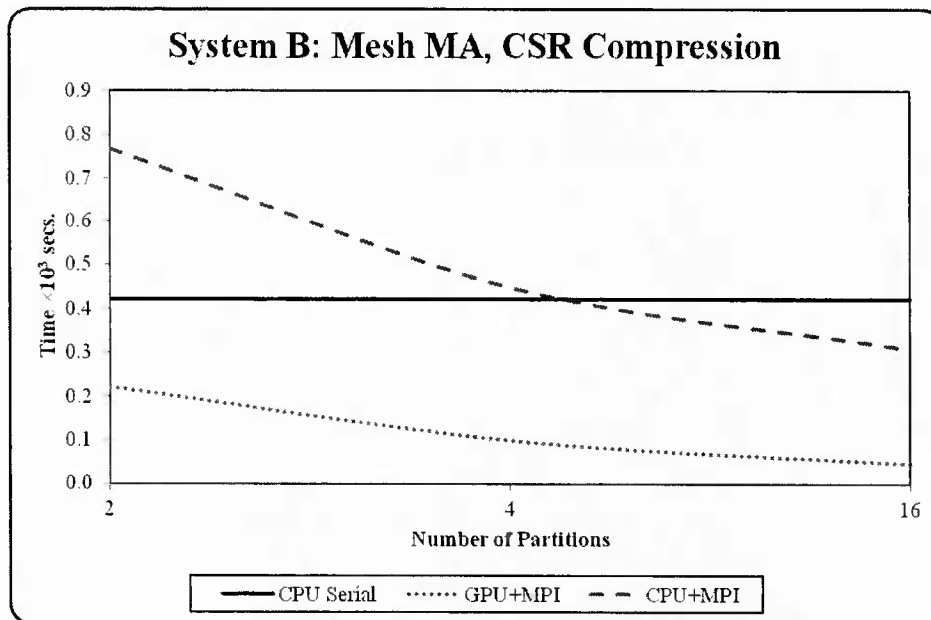


Figure 59. Multiple CPU/GPU computing system - mesh MA (System B).

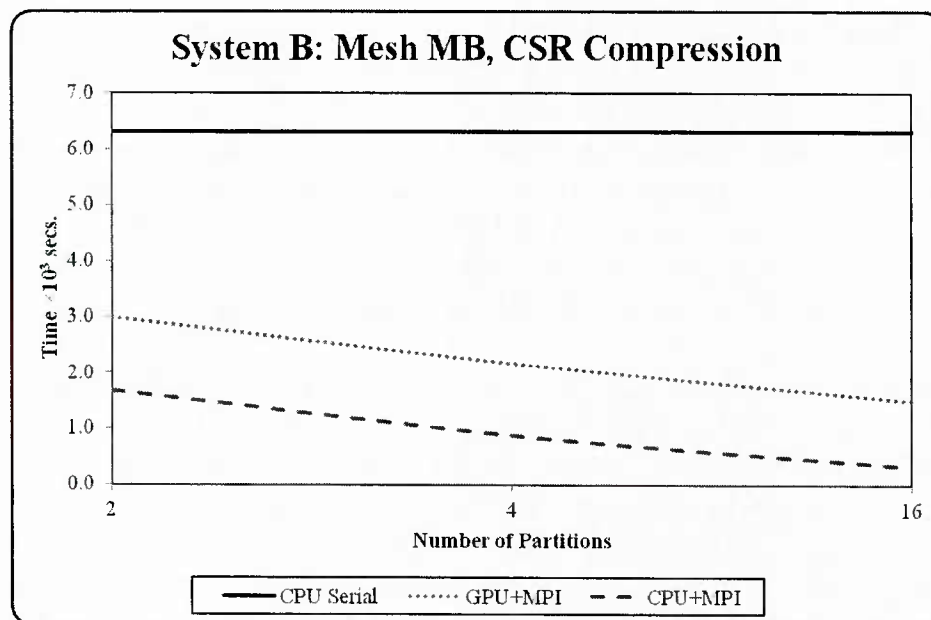


Figure 60. Multiple CPU/GPU computing system - mesh MB (System B).

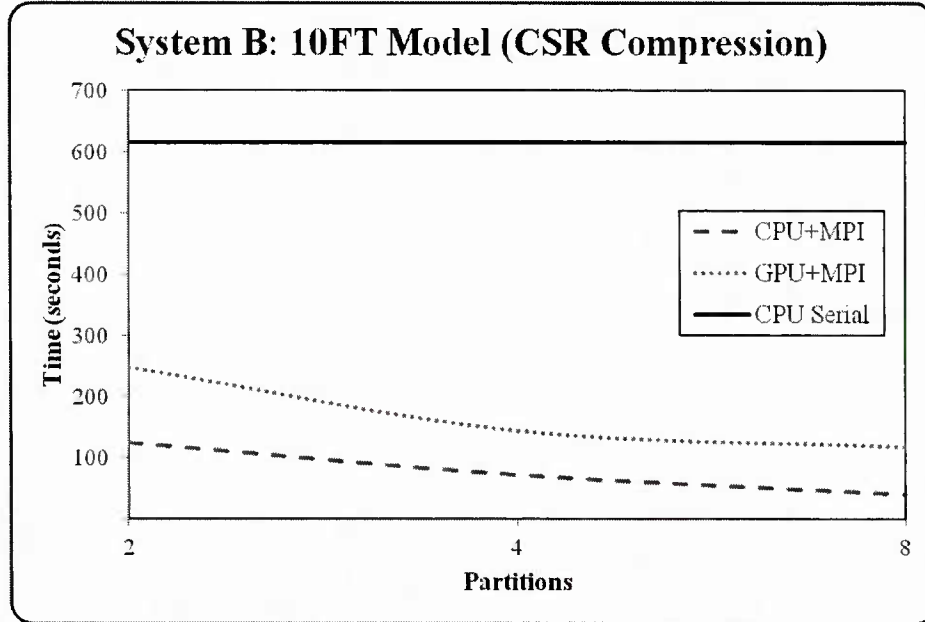


Figure 61. Multiple CPU/GPU computing system - mesh 10FT (System B).

5.3.3 Initial performance analysis. The execution of the unstructured mesh input files, **MA**, **MB** and **10FT** for both computing systems exposes some interesting performance behaviors, notably a divergence of performance for the different computing environments. The observed results for both **System A** and **System B** lower in total solution time as the number of sub-domains increases – excepting at 16 partitions, as **System A** then starts to ratchet up in solution times for both input meshes **MA** and **MB**. The input mesh **10FT** displays almost no difference between CPU plus MPI and GPU plus MPI – the impact of intra-nodal communication is lessened for this model within the **System A** computing system environment.

System B performs better using GPU plus MPI over CPU/MPI for the smaller input mesh **MA** but this is the opposite of that observed with the larger mesh **MB**. **System B** has both a larger set of registers and shared memory than **System A** and therefore able to hold larger amounts of data to increase throughput allowing better utilization and conversely exposing higher combined latencies of local CPU-GPU and intra-nodal communications. The result is the counter-intuitive effect of a less computationally intensive problem performing better than the larger and more computationally costly input mesh **MB** – an artifact of increased memory complexity and no direct connection to the MPI library calls.

The counter-intuitive effect of better hardware creating reverse performance for larger problem domains, i.e. less computationally intensive models can perform better than more computationally intensive models using a device that is optimal for systems requiring higher numbers of floating point operations can be seen in Figure 61. Figure 61 shows a nearly constant

difference from the GPU plus MPI and CPU plus MPI which is likely due to the more efficient execution of the GPU device – exposing a larger amount of intra-nodal communication cost for the global system.

The software factors that influence the performance of multiple CPU/GPU computing systems is discussed next followed by the corresponding hardware factors.

5.4 Software Data-Structures/Layout Factors

The previous section was an initial performance analysis for unstructured mesh inputs via the *multiple* CPU/GPU computing systems defined as **System A** and **System B** and produced mixed results – **System A** displayed a performance boost for the larger input mesh **MB** but not **MA** and **System B** displayed the converse. However neither approached the same level of performance observed by the single CPU/GPU systems of the previous chapter and neither illustrated definitive performance increase for input mesh **10FT** – although **System A** yields a closer result. The software variables involved in the observed results of the initial full solution with multiple CPU/GPU computing systems are examined to identify potential factors that can hinder performance of the presented candidate application. The first software factor to be examined is intrinsic to memory-bound problems such as the presented candidate composite process flow modeling finite element based application – data compression format.

In the interest of brevity, the reader is referred to **chapter 4** for a more detailed reasoning for the execution of the BCSR2x2 format over CSR which was defined for the initial performance results observed. The same system parameters that exist at the local *single* CPU/GPU computing systems are valid for the multiple CPU/GPU structure presented in this chapter, and the potential increase in locality via the utilization of the BCSR2x2 data compression format [52-54] is discussed next.

Figure 62 and Figure 63 show the multiple CPU/GPU performance with **System A** and **System B** respectively using only BCSR2x2 data compression format for input meshes **MA** and **MB**. The homogeneous comparisons of the BCSR2x2 show that **System A** gains no positive performance benefit using the smaller input mesh **MA** but does for the corresponding larger mesh **MB** once 16 partitions is reached whereas the CSR format showed that at 16 partitions the performance for this input mesh dropped. This difference in behavior corroborates the precept that locality defined at the software data layout can effect behavior of HPC applications.

Figure 66 and Figure 67 show the performance of the input mesh **10FT** for **System A** and **System B** respectively comparing the CSR and BCSR2x2 compression formats. The less regular mesh defined by the **10FT** model displays a consistent benefit when expressed using the

BCSR2x2 compression format over the CSR format. These observed performance results are consistent with the input mesh **MA**, which contains a similar number of non-zeros but in a much less complex geometry.

System B displays similar performance behavior as **System A** when the data compression layout is altered to BCSR2x2 but to a larger magnitude. **System A** showed performance benefit at 16 partitions for the input unstructured mesh **MB** whereas **System B** illustrates this same benefit at 4 partitions. And while **System A** has no discernible advantage of BCSR2x2 for the input mesh **MA**, **System B** does show a lower total solution cost, albeit not very impressive. *Table 30* and *Table 32* are the observed results utilizing the BCSR2x2 compression format for **System A** and **System B** respectively.

The observed results are taken within the context of the BCSR2x2 data compression formats only, with the base line defined as the cost of execution for the global solution using BCSR2x2 – i.e. the full solution cost using the BCSR2x2 compression format using a CPU/GPU computing system at a single processor level, with no domain decomposition applied. The effects of spatial locality are applied in a mixed compression environment next.

Table 31

Multiple CPU/GPU performance in seconds (System A)

Data Compression	Partitions	Mesh MA	Mesh MB
BCSR2x2	1	418.44	4,219.61
BCSR2x2	2	1,512.94	14,223.50
BCSR2x2	4	1,015.11	7,429.34
BCSR2x2	16	715.41	3,234.68

Table 32

Multiple CPU/GPU performance of 10FT model in seconds (System A)

Data Compression	Partitions	Mesh 10FT
BCSR2x2	1	140.69
BCSR2x2	2	1,020.35
BCSR2x2	4	660.38
BCSR2x2	8	551.57

Table 33

Multiple CPU/GPU performance in seconds (System B)

Data Compression	Partitions	Mesh MA	Mesh MB
BCSR2x2	1	217.76	1,931.31
BCSR2x2	2	233.34	2,151.81
BCSR2x2	4	168.96	1,202.46

BCSR2x2	16	129.29	549.83
----------------	----	--------	--------

Table 34

Multiple CPU/GPU performance of 10FT model in seconds (System B)

Data Compression	Partitions	Mesh 10FT
BCSR2x2	1	89.07
BCSR2x2	2	169.63
BCSR2x2	4	133.66
BCSR2x2	8	132.52

Figure 64 and Figure 65 show the comparison of the multiple CPU/GPU computing systems using different data formats of CSR and BCSR2x2 with **System A** and **System B** respectively. **System A** shows a positive benefit of using the BCSR2x2 format over the CSR format but only for a limited number of sub-domains for the input unstructured mesh **MB** and even less for the smaller input mesh **MA**. This observation illustrates that increasing spatial locality for the **System A** architecture can have absolute benefit as BCSR2x2 will improve on the CSR format and not just illustrate an ever increasing computational benefit as compared to a single compression format in all cases. **System B** does not follow the same pattern as **System A**.

The multiple CPU/GPU computing system defined by **System B** does not show any positive benefit for the use of the BCSR2x2 data compression format when direct comparisons to CSR are made – excepting a slight improvement for 2-partitions using the **MA** input likely due to *cache effects*. These observations are further elaborated in the next section on hardware factors as the immunity to the increased locality of **System B** when using BCSR2x2 is a consequence of this. *Table 34* and *Table 35* show the observed results for the comparison of CSR and BCSR2x2 data compression formats for **System A** and **System B** respectively. The less regular input mesh **10FT** shows a more consistent behavior for both computing environments.

Table 35

Multiple CPU/GPU performance in seconds – formats (System A)

Partitions	Mesh MA (CSR)	Mesh MA (BCSR2x2)	Mesh MB (CSR)	Mesh MB (BCSR2x2)
1	418.44	418.44	4,219.61	4,219.61
2	2,009.79	1,512.94	15,815.20	14,223.50
4	1,011.24	1,015.11	7,651.25	7,429.34
16	702.69	715.41	3,261.84	3,234.68

Table 36

Multiple CPU/GPU performance in seconds – formats (System B)

Partitions	Mesh MA (CSR)	Mesh MA (BCSR2x2)	Mesh MB (CSR)	Mesh MB (BCSR2x2)
1	168.57	217.76	1,197.35	1,931.31
2	335.92	233.34	2,138.79	2,151.81
4	157.90	168.96	1,038.69	1,202.46
16	124.02	129.29	495.33	549.83

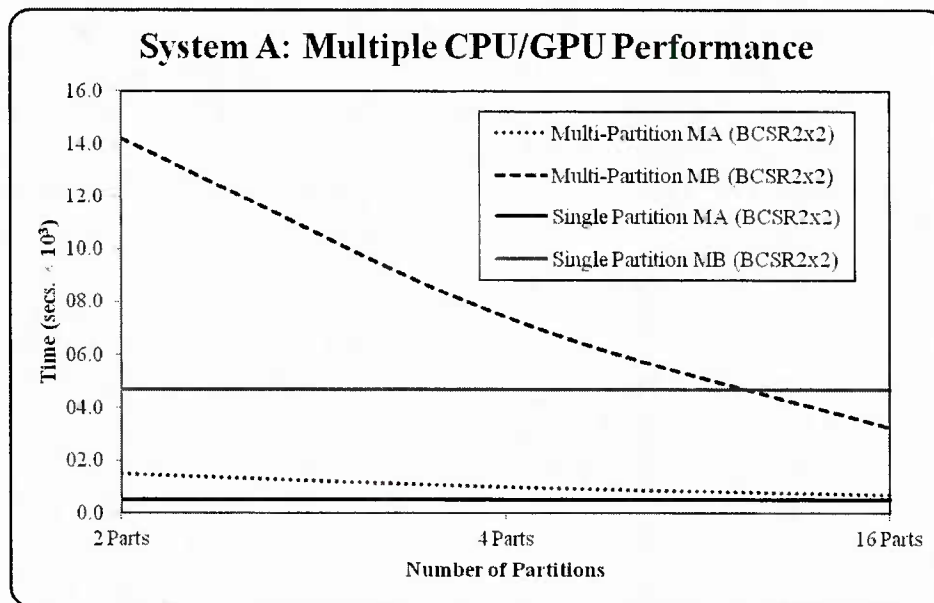


Figure 62. Multiple CPU/GPU performance BCSR2x2 compression (System A).

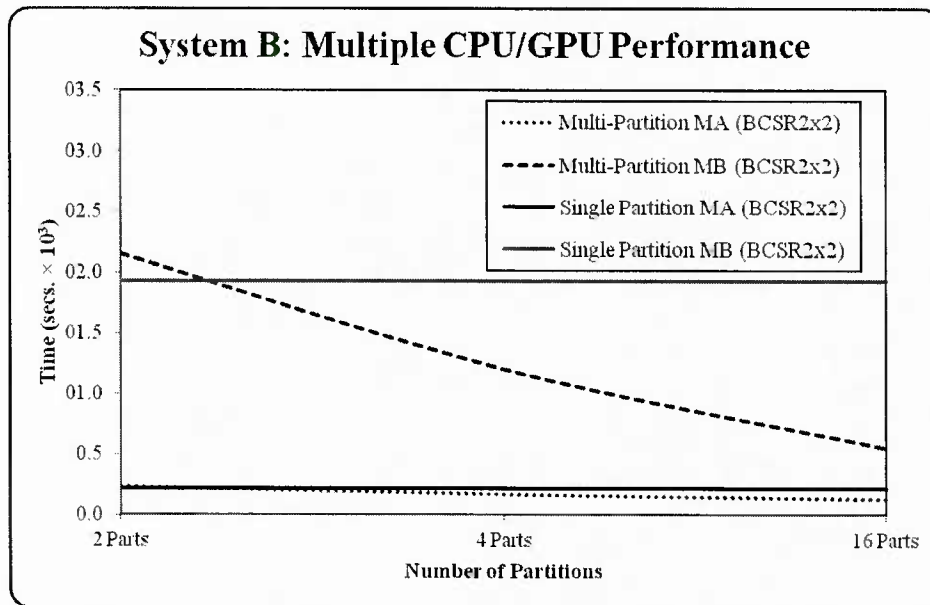


Figure 63. Multiple CPU/GPU performance BCSR2x2 compression (System B).

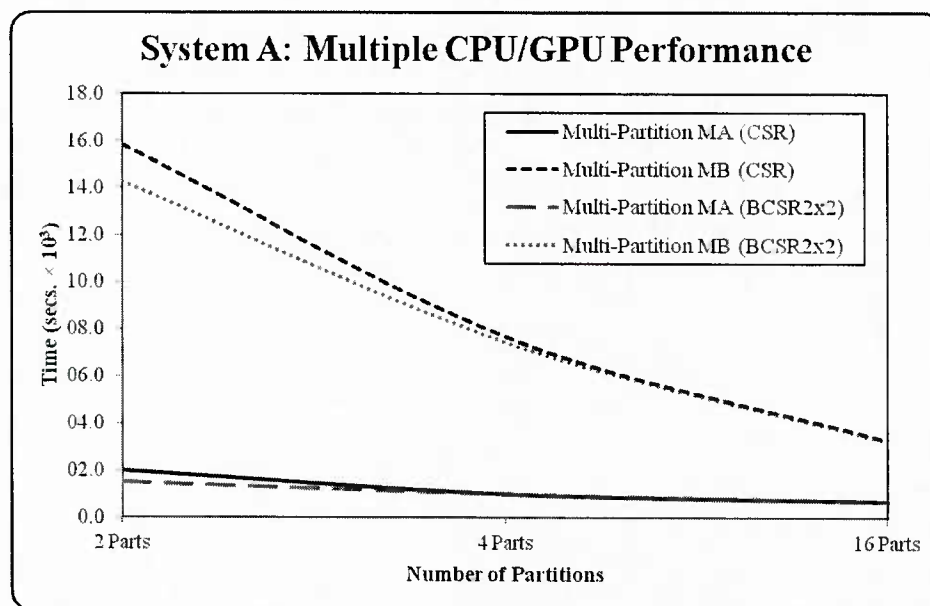


Figure 64. Multiple CPU/GPU performance mixed compression (System A).

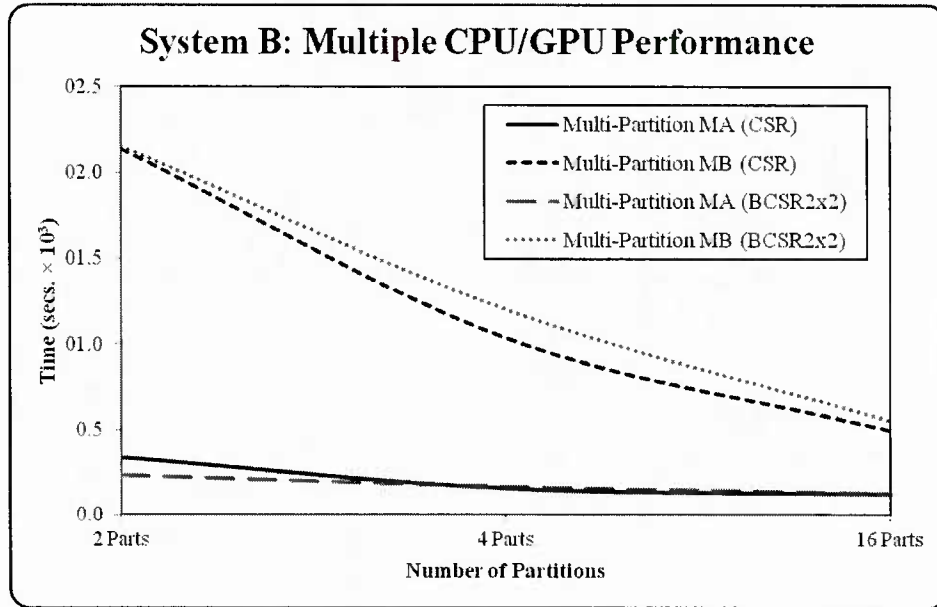


Figure 65. Multiple CPU/GPU performance mixed compression (**System B**).

Figure 66 and Figure 67 show the performance results for **System A** and **System B** respectively for the input mesh configuration **10FT**. The computing environments defined by **System A** and **System B** illustrate general equivalence of performance benefit for increased locality exposed by the use of BCSR2x2 – unlike the input meshes **MA** and **MB**. The more regular input meshes **MA** and **MB** have lower irregular memory access patterns than **10FT**, exposing hardware differences to a greater degree – improving locality for the algorithm has consistent results in both computing environments.

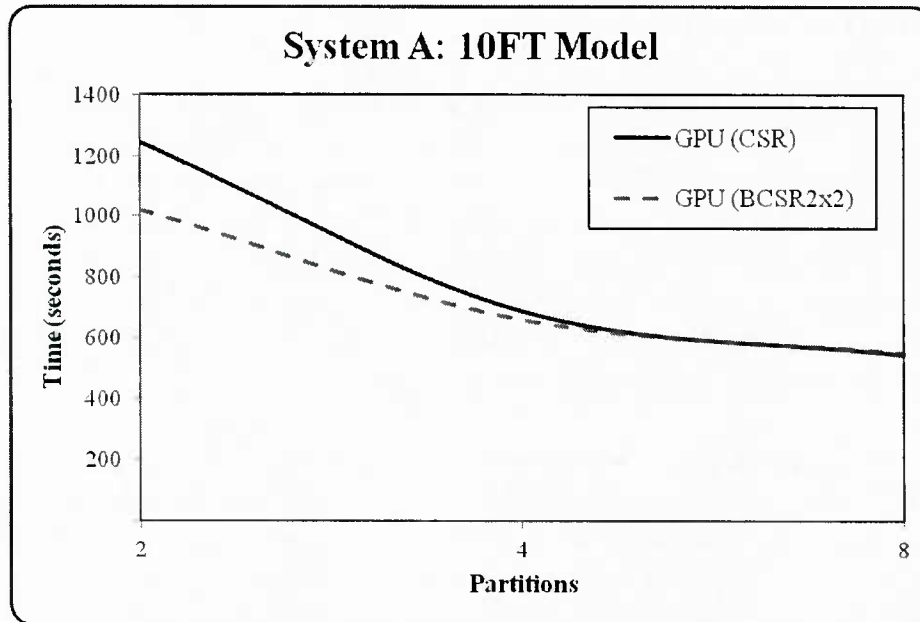


Figure 66. Multiple CPU/GPU performance mixed compression - 10FT (System A).

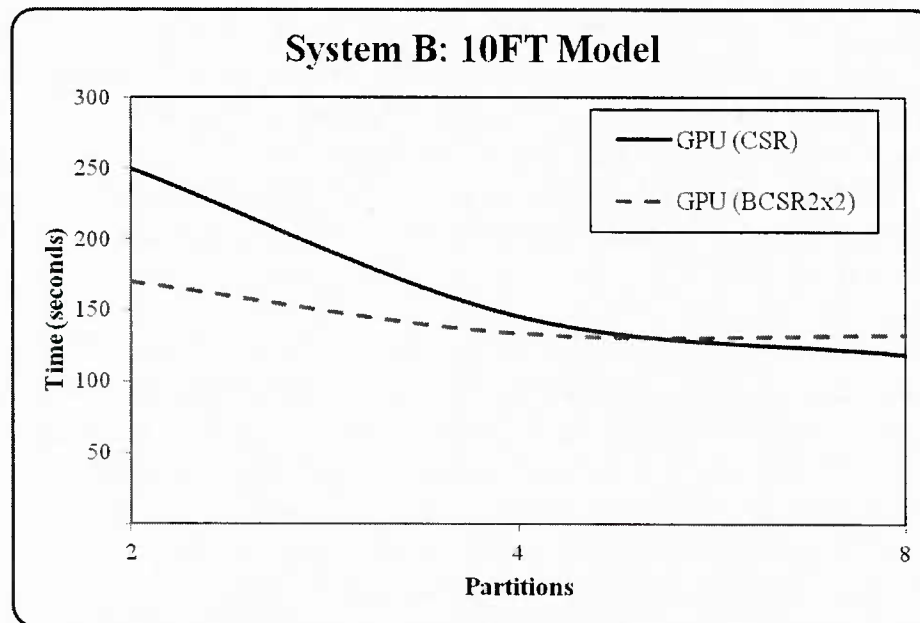


Figure 67. Multiple CPU/GPU performance mixed compression - 10FT (System B).

5.5 Hardware Architectural Factors

The observed performance of the CPU/GPU computing systems using the unstructured mesh input defined by **MA**, **MB**, and **10FT** is affected not just by the software factors discussed in the previous section but hardware factors as well. The presented candidate application performed with mixed results using the multiple CPU/GPU computing system paradigm, and switching to different data compression formats continued these amalgamated observations – providing enhanced results for **System A** but less so for **System B**. The lower sensitivity to the adjustment of data compression format of **System B** given the equivalence of partition counts and input meshes implies an underlying hardware factor.

The architectural design of **System B** is defined as CUDA compute architecture 2.0 and **System A** is defined as CUDA compute architecture 1.0 – significant architectural differences for these systems exist. CUDA's thread concept is register-bound and with **System B** embodying over 32,000 on-chip registers compared to **System A** with a little over 8,000 provides the ability of more resources for execution threads to remain viable – more importantly is the existence of an actual cache structure for **System B** that is absent from **System A**.

The inclusion of the **System B** cache and higher memory device I/O allow for higher throughput and a finer granularity than that provided by **System A**. This finer granularity and faster memory I/O for **System B** creates less sensitivity to the locality alterations provided via the BCSR2x2 data compression format, as the multiple CPU/GPU computing system has less problems with data locality than does **System A**. Therefore optimizing the performance of the presented candidate application has different requirements for the different architectures that need to be understood.

Figure 64 and Figure 65 exemplify the importance of a proper coordination of software and hardware factors for optimizing HPC applications. Figure 64 shows that **System A** is positively impacted with the application of BCSR2x2 due to a lack of hardware-level cache whereas utilizing BCSR2x2 to increase locality for **System B** is both unnecessary and potentially deleterious to performance as shown in Figure 65 using input meshes **MA** and **MB**. Increasing the number of elements in a single clock cycle with the implementation of BCSR2x2 using the multiple CPU/GPU computing system defined by **System B** is likely over-utilizing the on-chip hardware resources as competition increases.

The next section discusses the observed full solution performance using an augmented version of equation (4.9.3) from **chapter 4** such that the performance of multiple CPU/GPU computing systems is endorsed.

5.6 Computational Complexity Analysis

This section establishes the mapping of the observed performance and the derived complexity analysis for the *multiple* CPU/GPU computing system, detailed in Appendix A. The theoretical performance estimates for **System A** are discussed first followed by those for **System B** where all results are generated under the assumption of *CSR* data compression format.

The complexity analysis model for the multiple CPU/GPU systems is a natural extension from the previous chapter's derivation of the single CPU/GPU systems model – the results from the single analysis model are incorporated as a critical component of the multiple analysis model. However, the introduction of MPI as a communication amongst various sub-domains presents an added level of communication abstraction given that the GPU cannot communicate directly with the CPU it can neither communicate with the MPI library calls that can contain a significant amount of overhead [105]. Building from equation (4.9.3) in **chapter 4** and using value found for T_{gpu} , equation (5.3) with P_{sd} the number of sub-domains, N_{ATB} the number of active threads per block, and C the cost of combined combination of intra-node communication – assuming that P_{sd} is no greater than 16.

$$T_{mult_gpu} \equiv \frac{T_{gpu}}{P_{sd}} \left[\frac{C}{P_{sd}} (N_{ATB} + \sqrt{P_{sd}}) \right] e^{\frac{P_{sd}}{N_{ATB}}} \quad (5.3)$$

The determination of the performance modeling equation for *multiple* CPU/GPU systems is more involved than the *single* CPU/GPU system model - the individual architectures involved can present obfuscated operational costs which accentuate the PCIe *bottleneck* of the single system. Equation (5.3) depicts a change in computational cost when the number of sub-domains reaches 16 as CUDA waits until a *half-warp* is instantiated before issuing a memory transaction [37, 93] – this condition is meant to emulate this behavior across discrete systems.

The communication variable C from equation (5.3) is affected not only by the size of the problem domain but also individual architectures and MPI implementations involved and it is the *inter-play* of MPI and local CPU-GPU host communications that is generally deleterious to *multiple* CPU/GPU computing systems [105, 106, 110]. The understanding of how these communication factors interact with the determined modeling equation is discussed in the next sub-section.

5.6.1 Relationship of MPI-GPU and CPU-GPU communication. The Multiple CPU/GPU computing systems do not yield optimal parallel performance as expected when given

P processors and a sparse matrix with N_z total non-zero elements and R local GPU registers i.e. does not result in $\frac{1}{P}$ benefit [28]. The reason is that as the initial system of N_z elements is broken down into smaller sets that are held at the local GPU device yields more latency to hide and correspondingly less computational intensity to utilize GPU resources. This behavior was observed to be consistent across **System A** and **System B** for both input meshes when adjusted for local GPU register counts and associated N_z elements.

The expected behavior of a given computationally intensive application can be seen as related to the percentage of total N_z elements held locally at the GPU device and the number of partitions distributed across the global system. The *percentage* of total N_z elements held locally is given by equation (5.4) and was used as the independent variable to map the observed multiple CPU/GPU system performance against the optimal parallel behavior $\frac{1}{P}$.

$$\dot{R} \equiv \left(\frac{R \times P}{N_z} \right) \quad (5.4)$$

The ratio of the optimal parallel performance and actual performance for a given value of \dot{R} defines the *performance deviation from ideal* due to the local CPU-GPU host and MPI communication inter-play. These *deviations* were mapped using regression such that NZ_{local} is the ratio of the total number of non-zero elements, N_z , from the global problem domain held by the local GPU device and \hat{X}_p is the value of the deviation computed as the *ratio* of ideal parallelism $\frac{T_s}{P}$ and the *actual* execution time for the given number of partitions P and execution time for the *serial* version T_s represented as the solid BLACK lines in Figure 68, Figure 69, Figure 70 and Figure 71. These are shown for **System A** as Figure 68 and Figure 69 for input meshes **MA** and **MB** respectively - Figure 70 and Figure 71 illustrate these same factors for input meshes **MA** and **MB** using **System B**.

The equations revealed by regression, shown for convenience in Figure 68, Figure 69, Figure 70, and Figure 71 as the dashed RED lines, vary with (5.4) as input but can be easily replaced by the number of processors P as the proportion of N_z elements held by a local GPU device is directly related – this same precept holds for input mesh **10FT**. The equations derived via regression have a *Pearson Product-Moment correlation coefficient* of 1 in all cases, the coefficient dependency is one-to-one [111] – i.e., exact match with the *deviation from ideal* represented as \hat{X}_p in the figures. The approximated equations derived with regression from \hat{X}_p

are *degree 3* for all models and essentially isolate the overhead of CPU-GPU local host and intra-nodal communication costs. These approximated polynomial equations are employed as an asymptotic measurement. Therefore, by the definition of asymptotic behavior [91], the relationship of MPI and local CPU-GPU communication effects on *multiple* CPU/GPU computing systems can be shown as (5.5).

$$O(\hat{R}^3) \quad (5.5)$$

The equations derived via *regression* are specific to the observed performance for a given input mesh and architecture, but extending the number of partitions i.e. increasing independent variable against equation (5.5) and applying some constant K define the cost of intra-nodal and local CPU-GPU host communication will not be greater than *cubic*.

The asymptotic equation (5.5) is compared to the equations that were derived using regression for both **System A** and **System B**. Figure 72 and Figure 73 show the asymptotic behavior of the input meshes **MA** and **MB** for **System A** respectively and Figure 74 Figure 75 show input meshes **MA** and **MB** for **System B**. The results are shown in Figure 76 and Figure 77 for input mesh **10FT** using computing **System A** and **System B** respectively. These figures show that regardless of the number of partitions/sub-domains the theoretical cost of local CPU-GPU and intra-nodal communication, represented as the solid line, indeed stay below *cubic*, represented as the dashed line, for all models and both computing system environments.

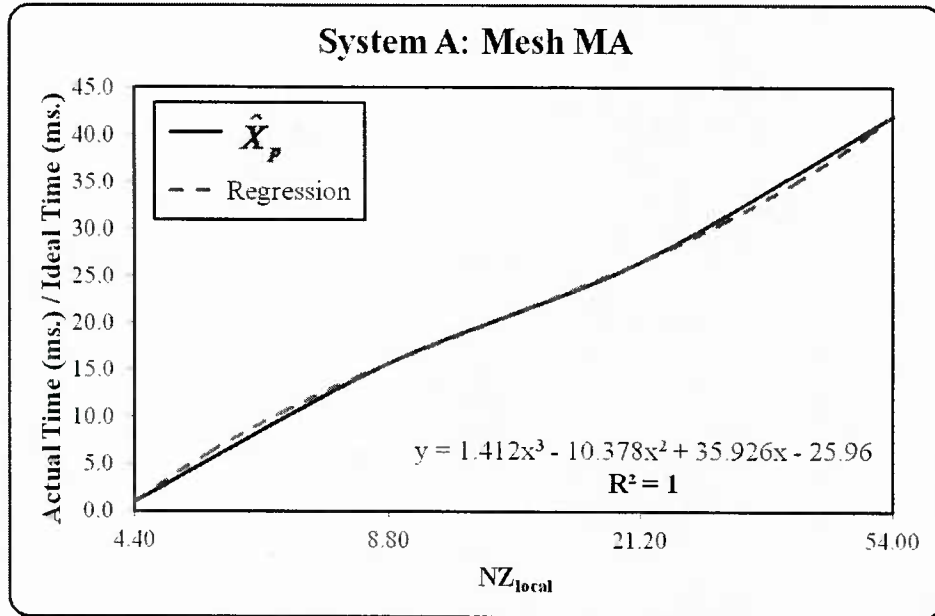


Figure 68. Deviation from ideal mapped with regression *multiple* CPU/GPU mesh **MA**.

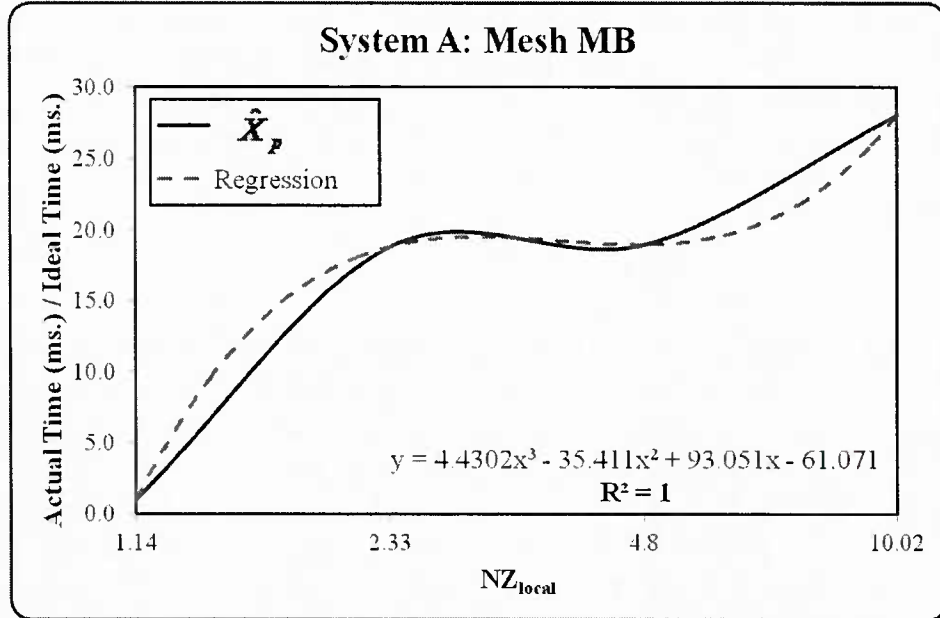


Figure 69. Deviation from ideal mapped with regression *multiple* CPU/GPU mesh **MB**.

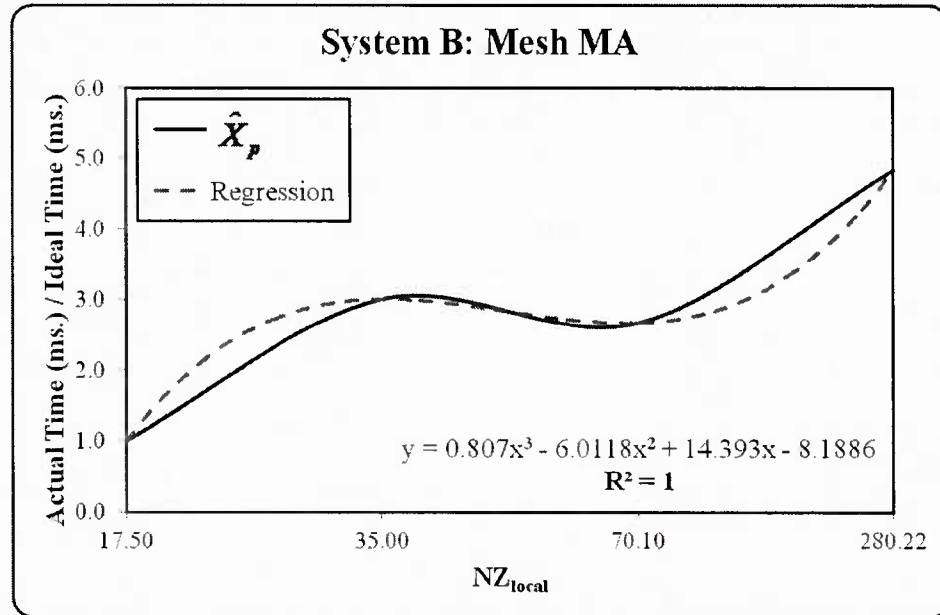


Figure 70. Non-zeros held locally (mesh **MA**) and factors off with multiple CPU/GPU.

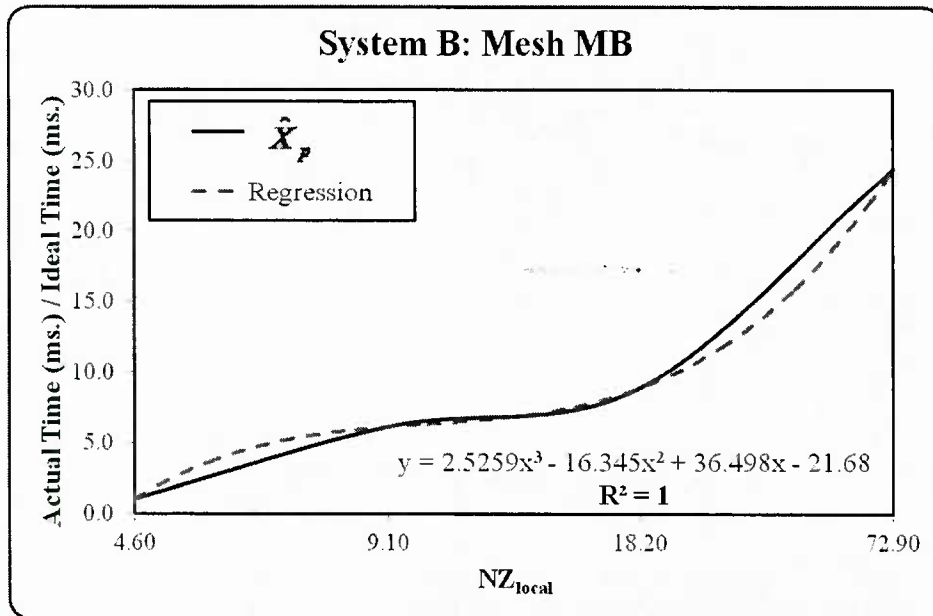


Figure 71. Non-zeros held locally (mesh **MB**) and factors with multiple CPU/GPU.

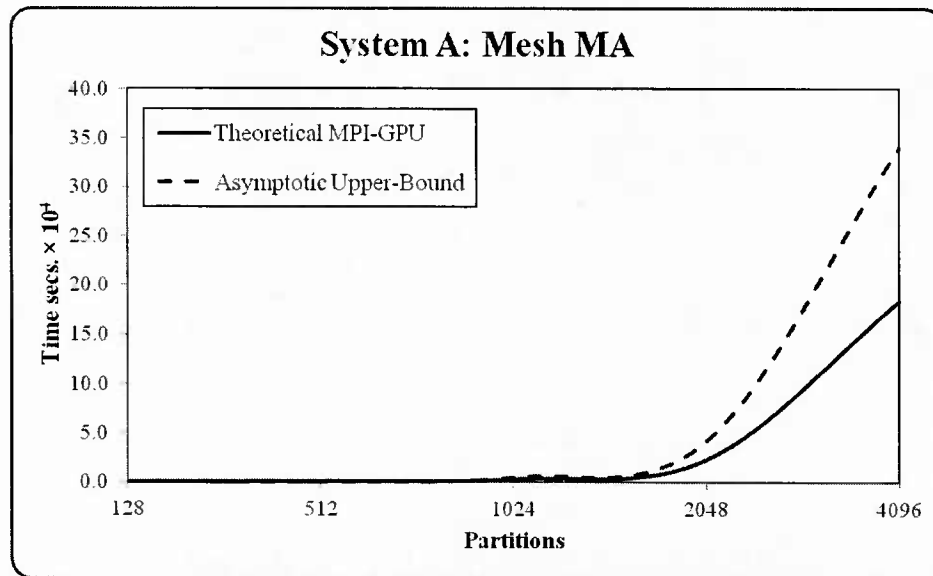


Figure 72. Asymptotic behavior of MPI and CPU-GPU communication (**MA**, System A).

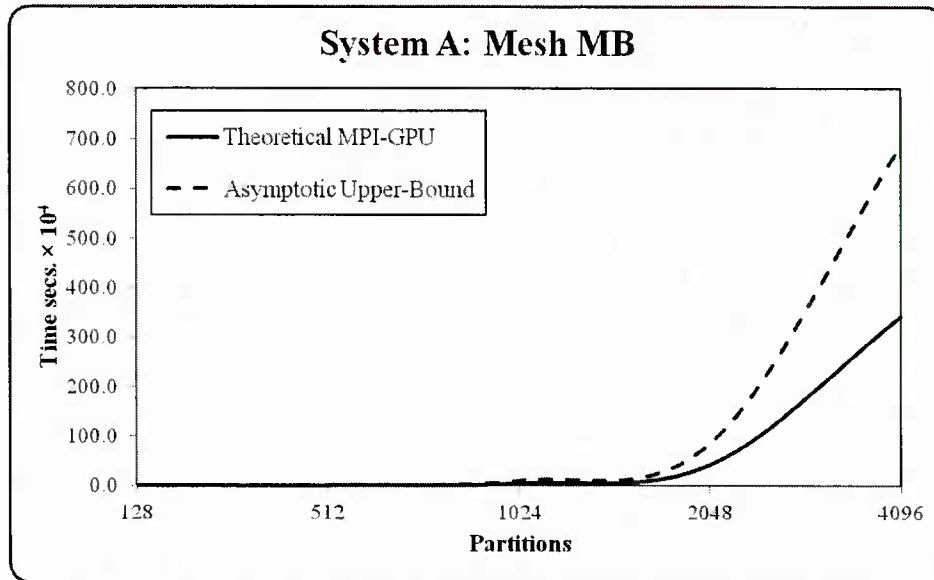


Figure 73. Asymptotic behavior of MPI and CPU-GPU communication (MB, System A).

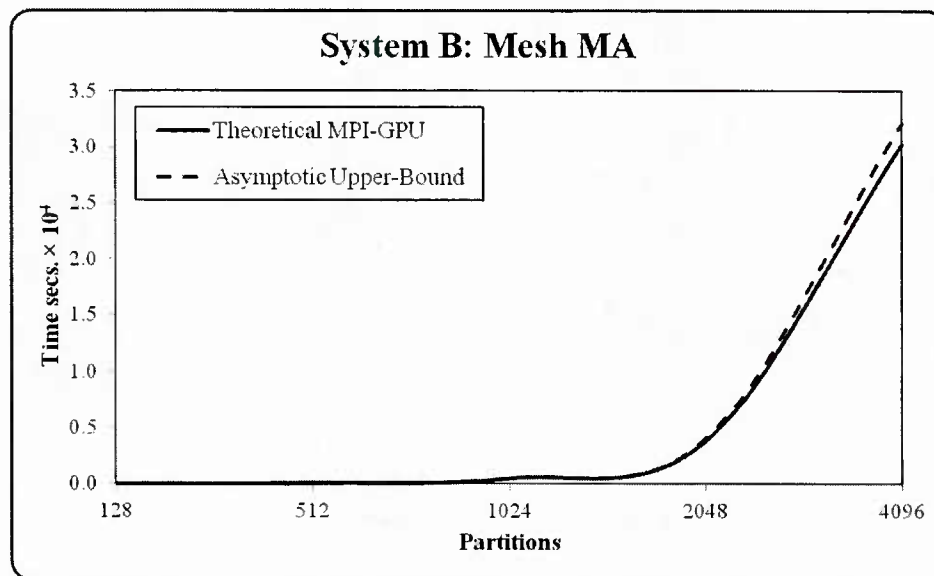


Figure 74. Asymptotic behavior of MPI and CPU-GPU communication (MA, System B).

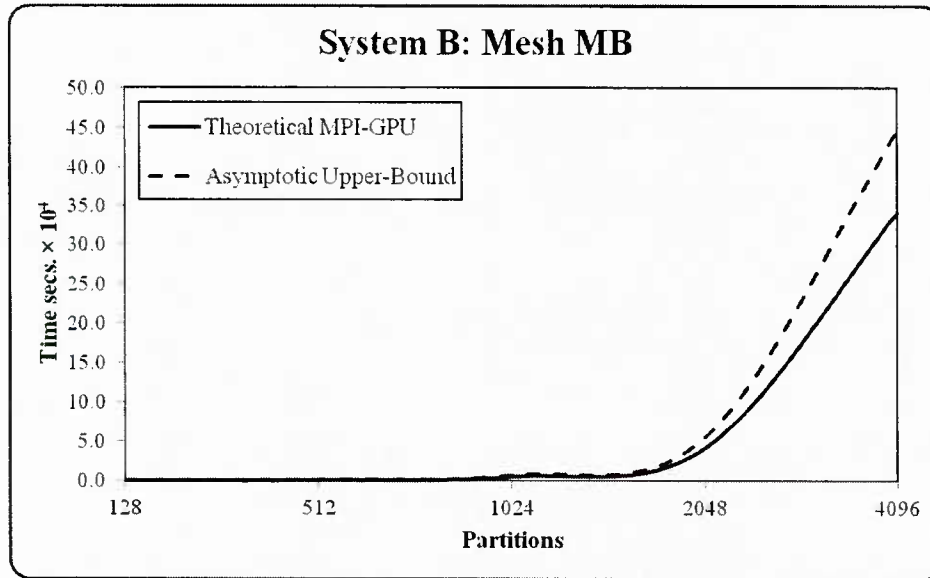


Figure 75. Asymptotic behavior of MPI and CPU-GPU communication (MB, System B).

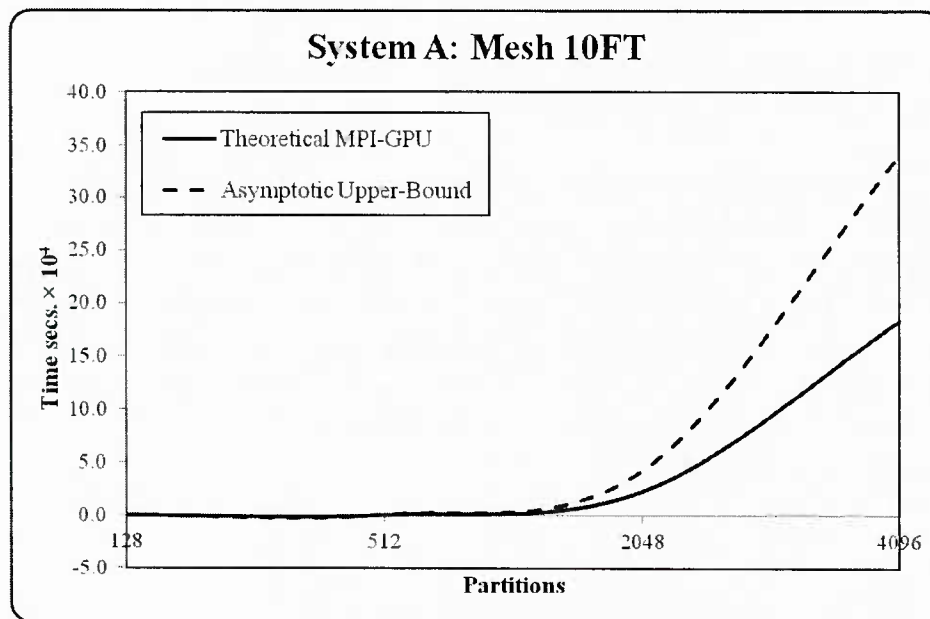


Figure 76. Asymptotic behavior of MPI and CPU-GPU communication (10FT, System A).

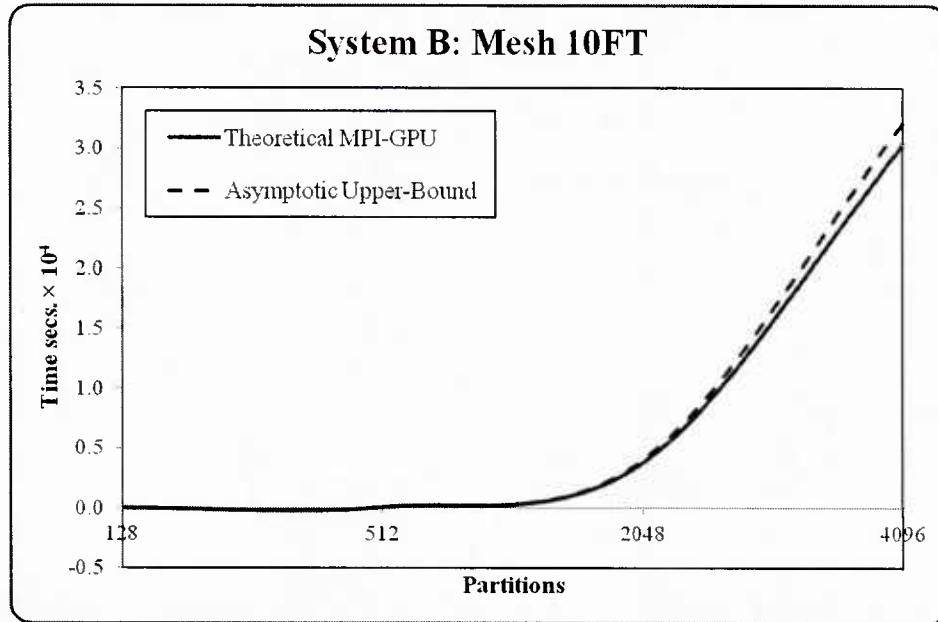


Figure 77. Asymptotic behavior of MPI and CPU-GPU communication (10FT, System B).

The next sub-section examines the theoretical performance results using the model given by equation (5.3) and altering hardware factors.

5.6.2 Comparison of performance modeling. The Multiple CPU/GPU computing system performance predictive model is compared to the actual time for each sub-domains for both **System A** and **System B** for input meshes **MA** and **MB** with the same input parameters. Figure 78, Figure 79 and Figure 80 show a strong correlation to the modeled performance given by equation (5.3) and the actual full solution execution time for the multiple CPU/GPU system defined by **System A**. Figure 81, Figure 82 and Figure 83 show a strong correlation to the modeled performance equation (5.3) and the actual full solution execution time for the multiple CPU/GPU system by **System B**.

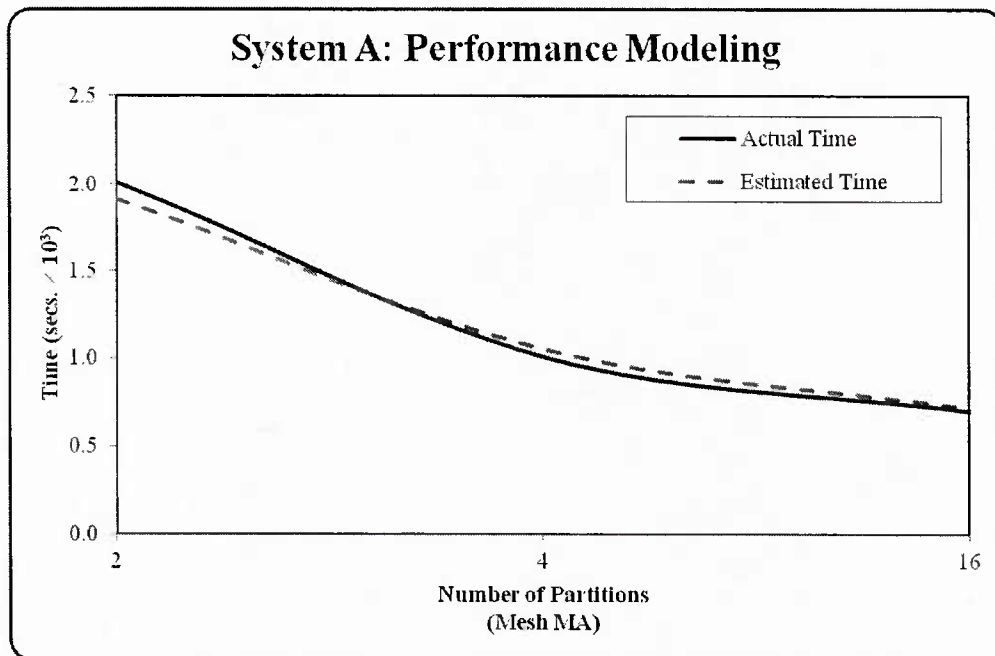


Figure 78. Multiple CPU/GPU theoretical performance with input MA (System A).

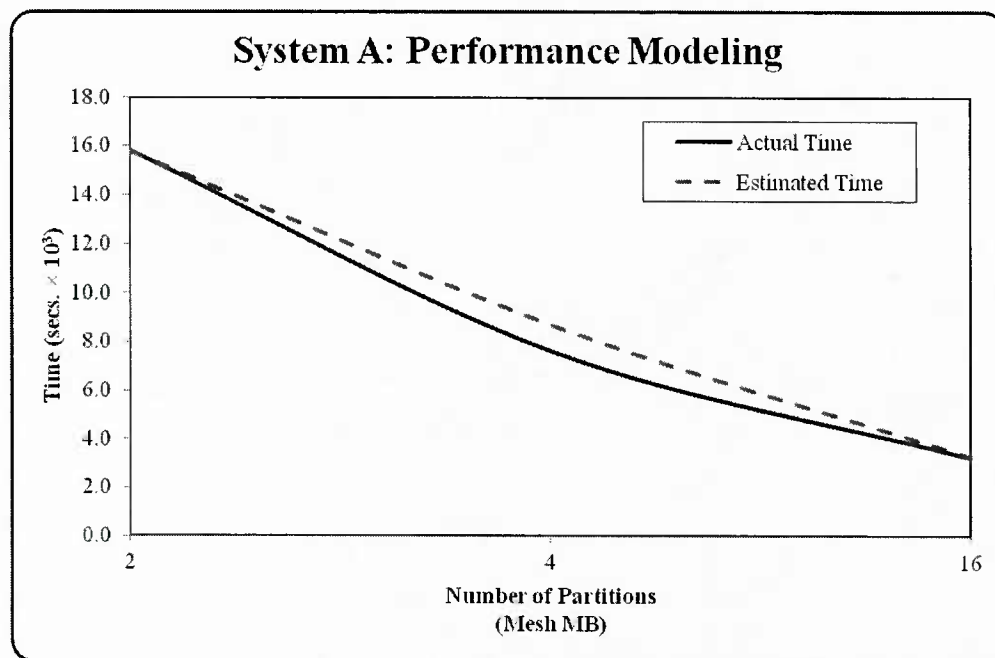


Figure 79. Multiple CPU/GPU theoretical performance with input MB (System A).

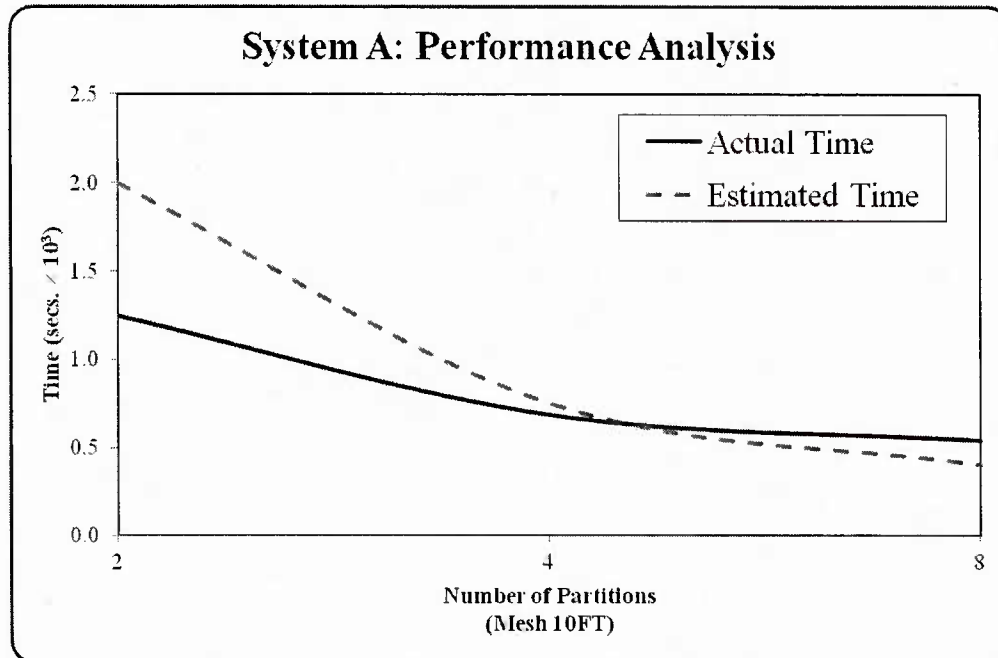


Figure 80. Multiple CPU/GPU theoretical performance with input 10FT (System A).

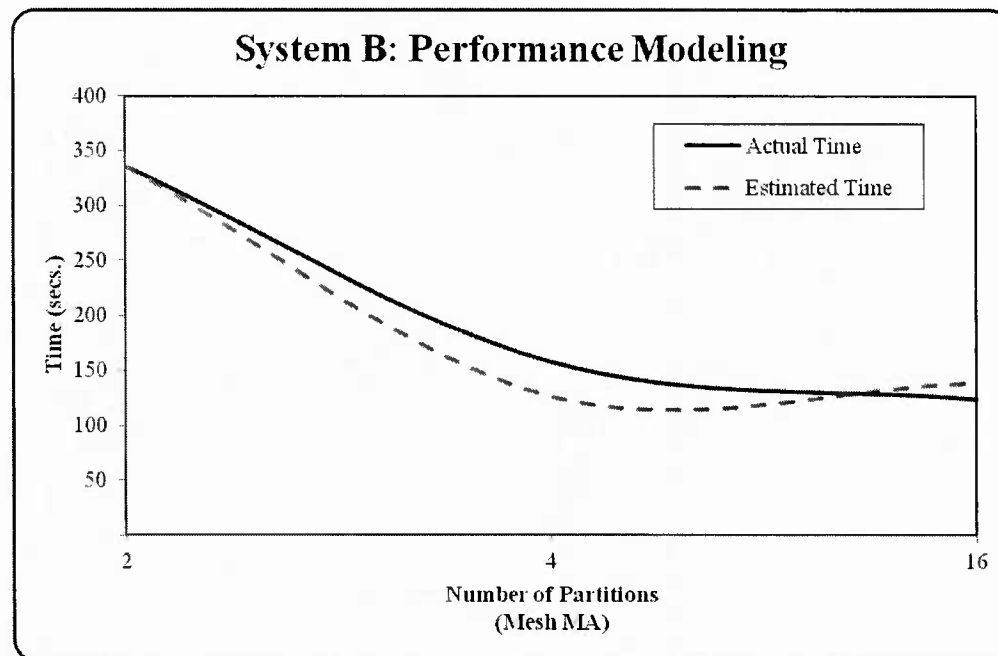


Figure 81. Multiple CPU/GPU theoretical performance with input MA (System B).

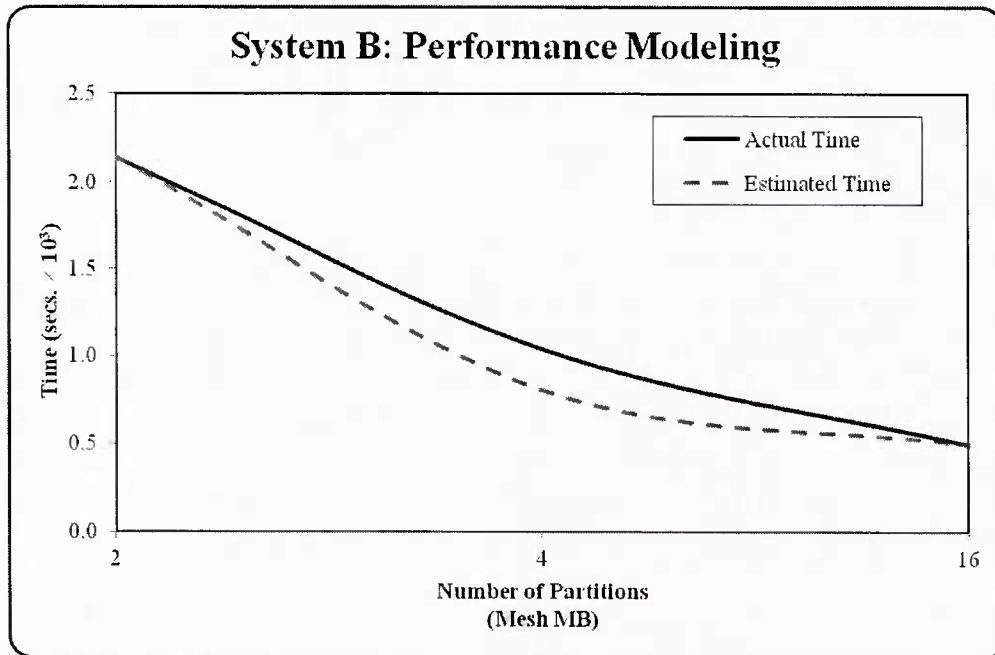


Figure 82. Multiple CPU/GPU theoretical performance with input MB (System B).

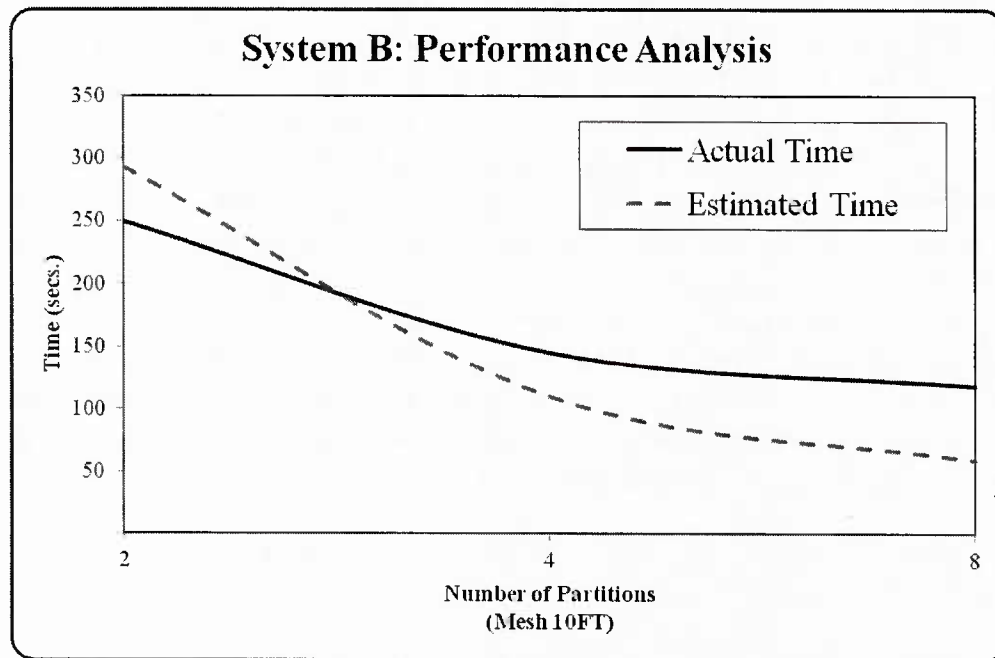


Figure 83. Multiple CPU/GPU theoretical performance with input 10FT (System B).

5.6.3 Contribution of Hardware Factors. This section establishes a relationship to hardware factors and the resulting application performance via the derived equation (5.3),

adjusting hardware variables and then projecting against the actual performance of the application. The resulting differentials are analyzed and the impact of the adjusted parameter(s) on performance of the CPU/GPU computing system is theorized. The number of SMPs for each of the defined computing systems is adjusted while the rest of the model is held as constant to isolate the specific hardware.

Figure 84 and Figure 85 show that as the number of SMPs drops the corresponding theoretical performance decreases for **System A** for both input mesh **MA** and **MB** respectively – due to the lower computational power of the individual processing elements. Increasing the number of SMPs for **System A** has the opposite effect on theoretical performance for both input mesh **MA** and **MB**, directly related to the greater computational power that is leveraged at this alteration. These theoretical performance results are consistent for Figure 84 and Figure 85 with the decrease of SMPs, shown as the dashed RED lines, producing greater effect when compared to the corresponding increase of SMPs shown as the dotted BLUE lines.

Figure 86 shows the less structured input mesh **10FT** and depicts a theoretical behavior across increasing partitions/sub-domains as roughly reflective of that for the more structured input meshes **MA** and **MB** for **System A** shown in Figure 84 and Figure 85. Increasing the number of SMPs will lower the total execution time and decreasing the number of SMPs will raise the total execution time. However, the degree of change is significantly higher for the **10FT** mesh using **System A** – likely due to the more distributed nature of the mesh, generating a correspondingly less regular sparse matrix and coercing more indirection in the data compression format. The increased indirection of the sparse matrix-vector multiplication for the **10FT** model combined with lower SMPs means lower process throughput hindered by higher levels of irregular memory access patterns, significantly deteriorating latency hiding.

Figure 87 and Figure 88 show that **System B** has a similar theoretical behavior to that produced by **System A**; however the theoretical performance is much less pronounced. Theoretical performance drops for both increasing and decreasing the number of SMPs at 4 partitions as the computational intensity becomes *less salient* and the communication costs for intra-nodal communication overtake the final results.

The next sub-section discusses the software factors on theoretical performance for *multiple* CPU/GPU computing systems.

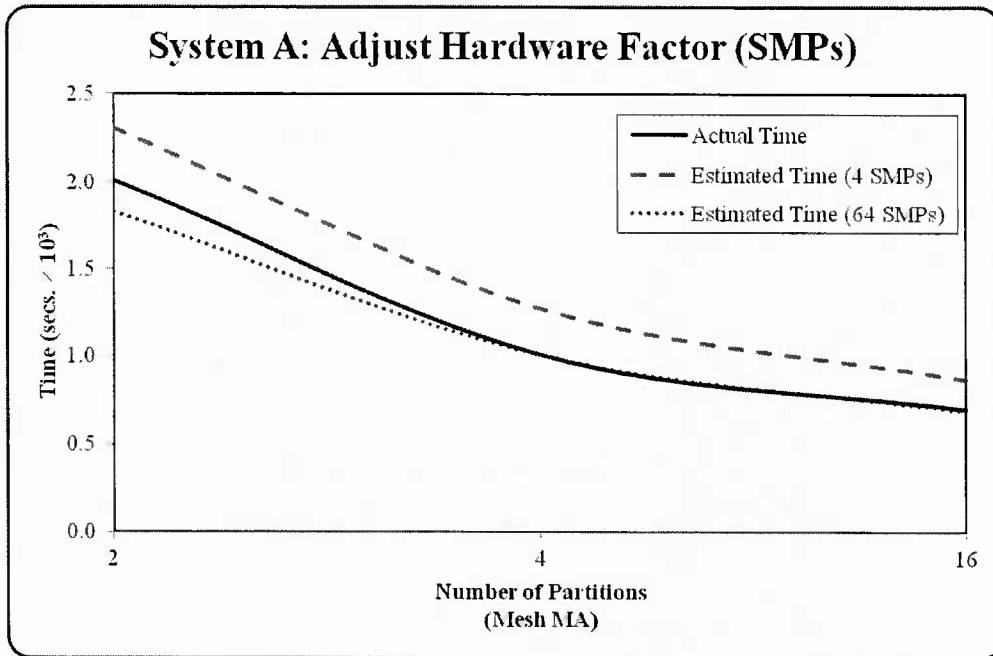


Figure 84. Multiple CPU/GPU performance with MA - hardware factor (System A).

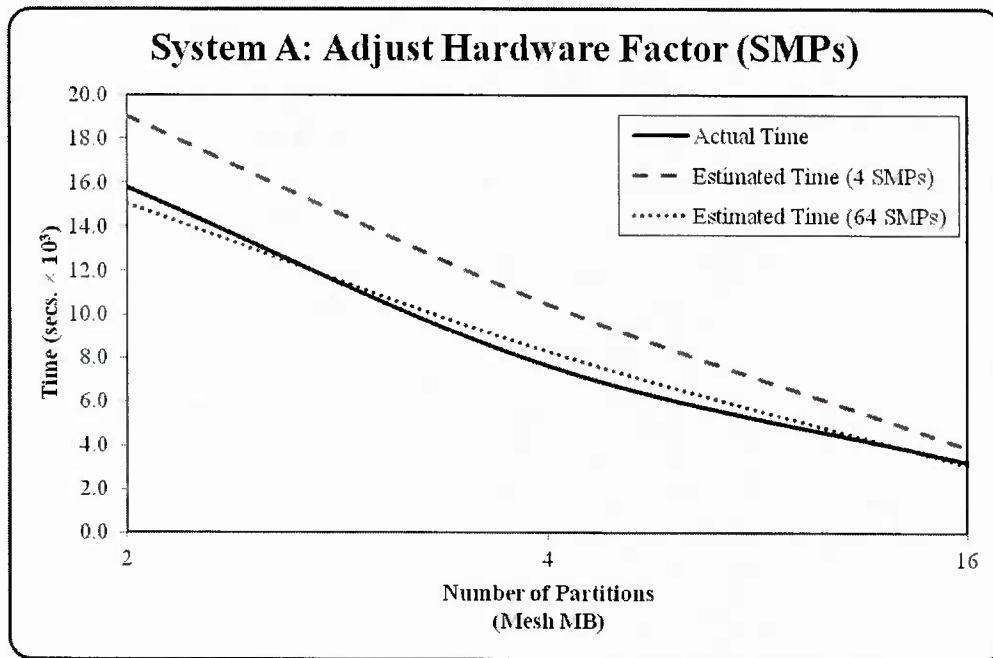


Figure 85. Multiple CPU/GPU performance with MB - hardware factor (System A).

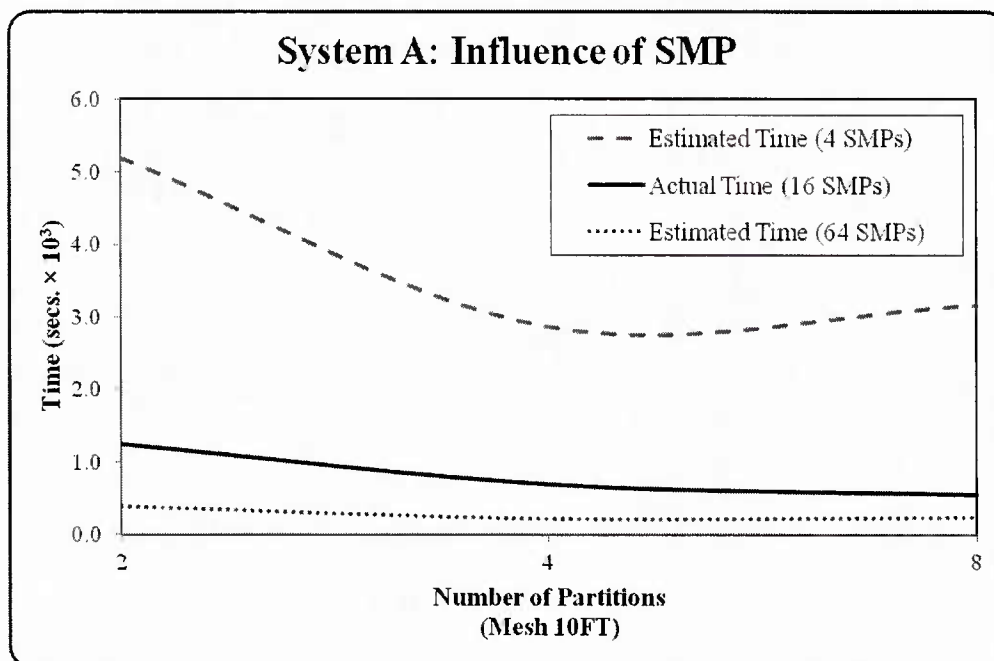


Figure 86. Multiple CPU/GPU performance with 10FT - hardware factor (System A).

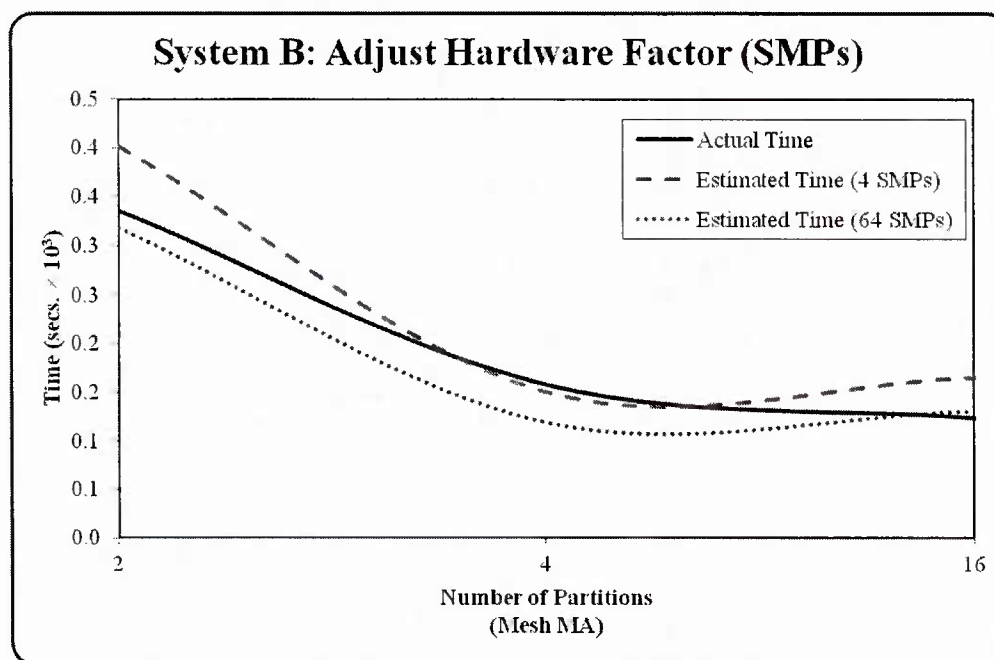


Figure 87. Multiple CPU/GPU performance with MA - hardware factor (System B).

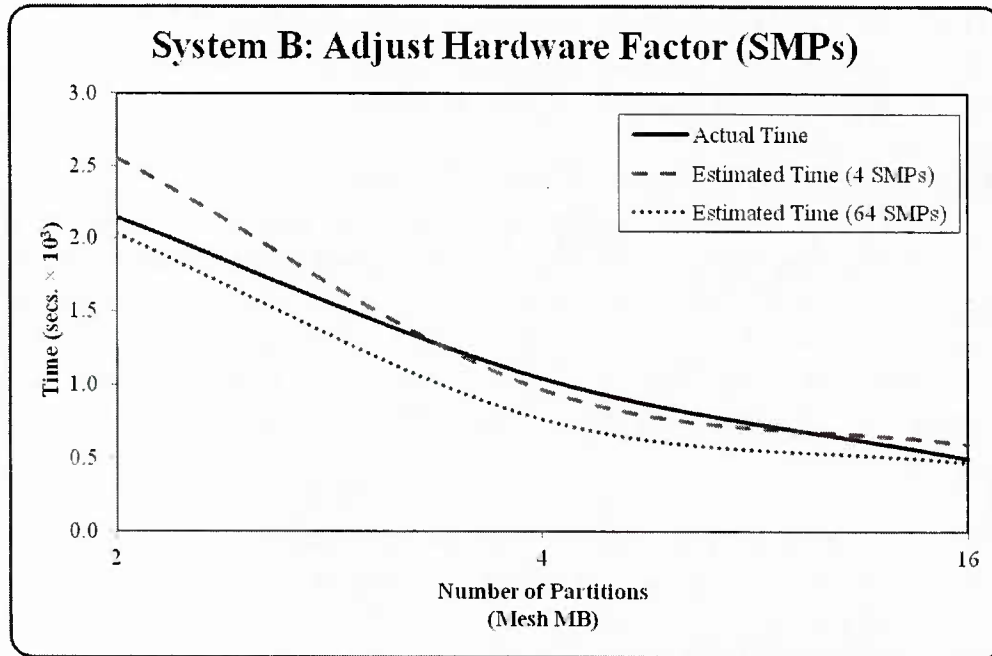


Figure 88. Multiple CPU/GPU performance with MB - hardware factor (System B).

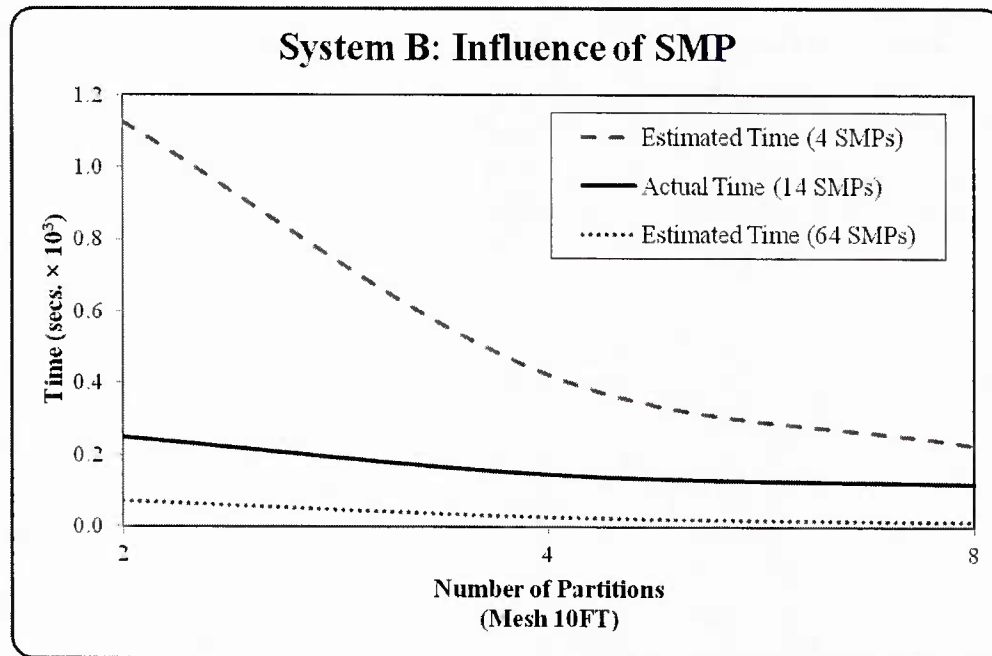


Figure 89. Multiple CPU/GPU performance with 10FT - hardware factor (System B).

5.6.4 Contribution of Software Factors. This section establishes a relationship to software factors and the resulting application performance via the derived equation (5.3),

adjusting hardware variables and then projecting against the actual performance of the application. The resulting differentials are analyzed and the impact of the adjusted parameter(s) on performance of the CPU/GPU computing system is theorized.

Thread occupancy is a common practice for increasing the performance of GPU-based systems in the GPGPU computing community and this paradigm is followed to achieve theoretical performance boost, altering the number of **Threads-Per-Block** (TPB). Increasing the **TPB** value in equation (4.9.3) is carried through to equation (5.3) and allows higher probability of coalesced memory accesses and utilizes more floating-point operational units – e.g. improved theoretical performance. Once again, **System A** displays the clearest benefits for both input meshes as shown in Figure 90 and Figure 91 - Figure 92 clearly illustrates the best performance at 256 threads per block.

Lower the number of TPB to less than optimal for **System A**, defined as 256, provides less opportunity for address coalescing as well as lower throughput whereas increasing the **TPB** has the converse theoretical effect. Figure 93, Figure 94 and Figure 95 for **System B** display similar effects on theoretical performance observed on System A excepting the sudden “dip” encountered at 4 partitions for both input meshes.

The observed theoretical “dip” at 4 partitions for **System B** is an artifact of equation (4.7) with the cost of the denominator \sqrt{B} . **System B** with input mesh **MB** shows a lowering of the effects of both higher and lower **TPB** as the number of partitions increase due to the growing influence of intra-nodal communication latency as well as increasing calls to the local CPU via the PCIe bus for each node in the global system.

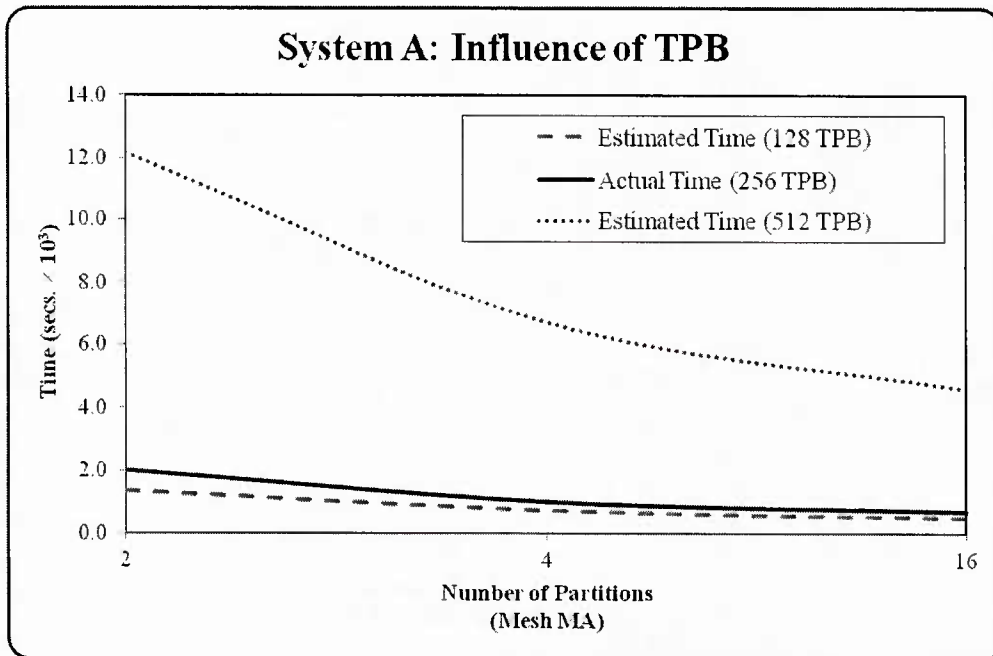


Figure 90. Multiple CPU/GPU performance with **MA** - software factor (System A).

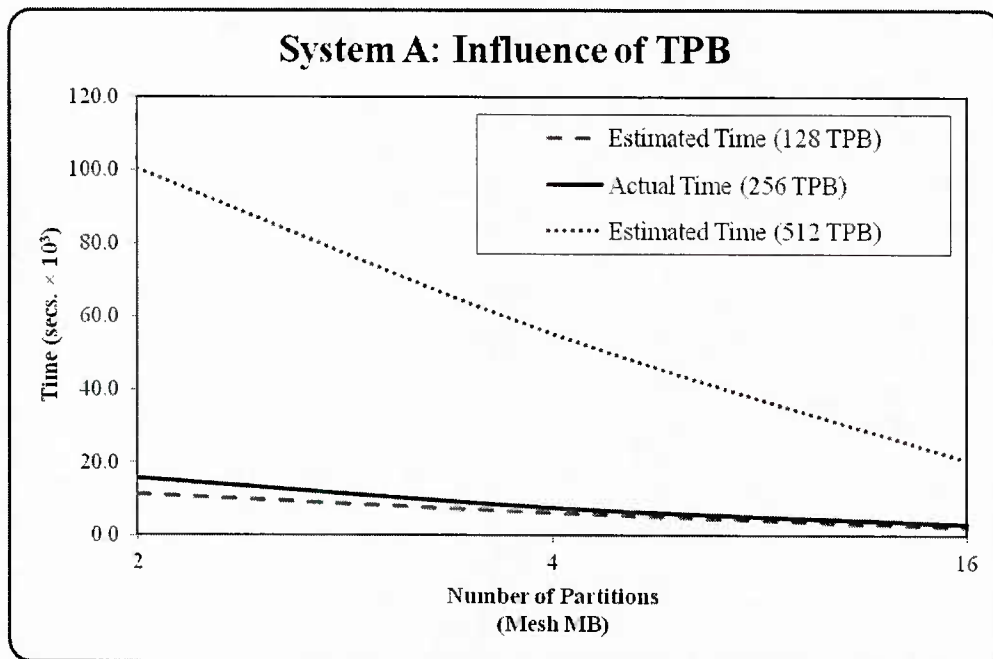


Figure 91. Multiple CPU/GPU performance with **MB** - software factor (System A).

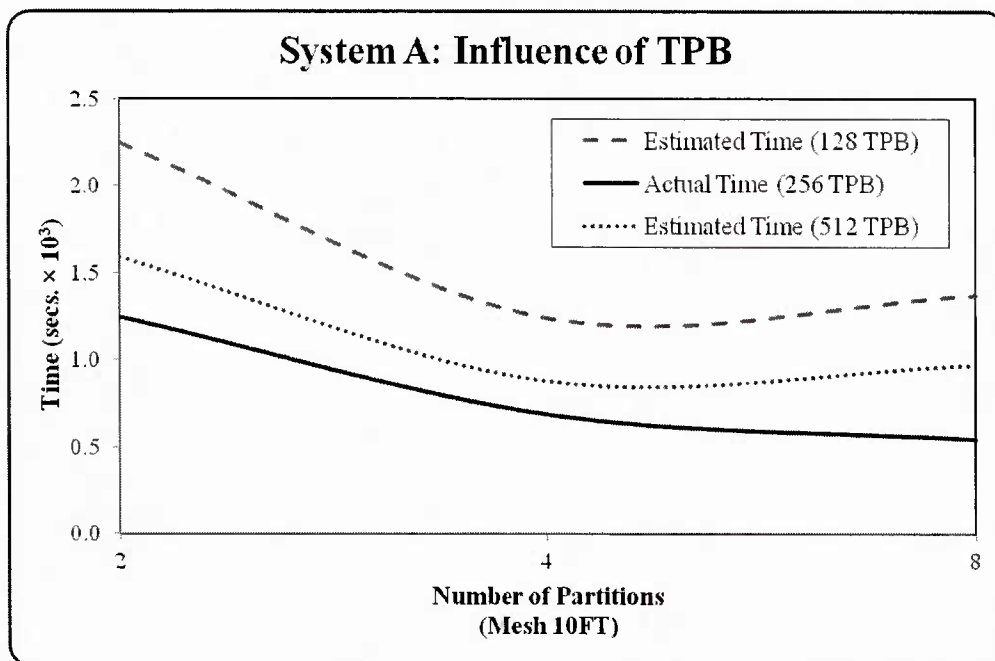


Figure 92. Multiple CPU/GPU performance with 10FT - software factor (System A).

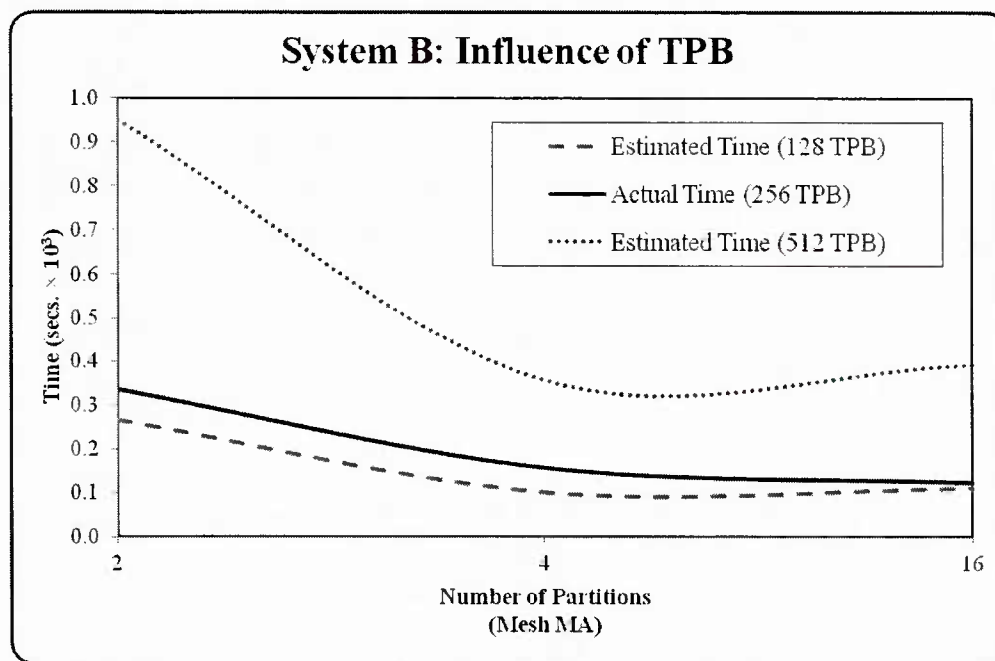


Figure 93. Multiple CPU/GPU performance with MA - software factor (System B).

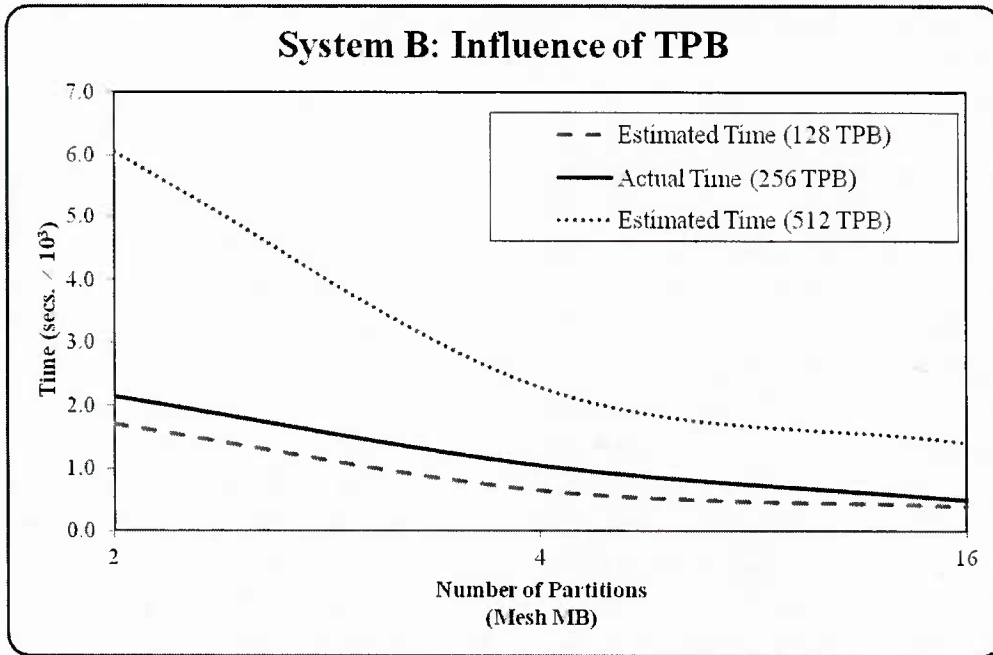


Figure 94. Multiple CPU/GPU performance with MB - software factor (System B).

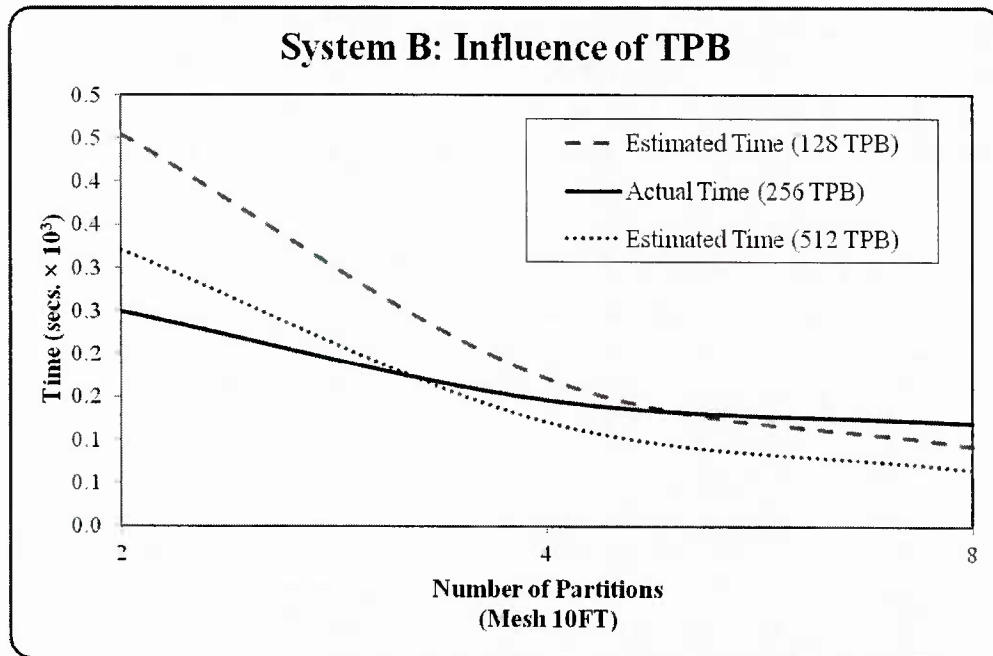


Figure 95. Multiple CPU/GPU performance with 10FT - software factor (System B).

The next section relates the observations of hardware and software factors to the final performance results of the presented candidate application, reasoning the importance of careful

use for multiple CPU/GPU computing systems for optimal HPC modeling application performance.

5.7 Performance and Relation to Software and Hardware Factors

The resulting performance of the multiple CPU/GPU computing system is *directly* tied to the interplay of software and hardware factors of the environments in which they executed and are related in this section – simply expanding hardware chips will not necessarily produce the desired performance boost if the algorithm poorly incorporates the hardware and vice versa. Point in fact, just loading a system with the largest possible number of threads (a software factor) will overload the register file (a hardware factor) with resource demands enforcing less utilization as well as register spilling to device memory and increasing the number of clock cycles to hundreds. Increasing the number of computational chips via the increasing number of SMPs (a hardware factor) will mean little if the access pattern of a matrix system defined by the application (an algorithmic factor) is accessed by Kernel threads in a row-major order when the GPU device is optimized for column-major causing non-contiguous addressing.

The *single* CPU/GPU systems from the previous chapter illustrate the overlap of software and hardware artifacts on resulting performance and the *multiple* CPU/GPU systems in this chapter show the same influence. However, this is not as easy to spot as the aggregate costs imposed by intra-node communication can abrogate any performance benefits observed. And determining the cost of this intra-node communication is difficult given its combination of the local PCIe overhead of CPU/GPU communications.

The current state of the GPU is one of isolation from the CPU as well as the MPI standards – this is an area of current research and concern for future co-processor accelerators [36, 96, 97, 112, 113]. Equation (5.3) takes liberties and employs approximation with regard to the final cost of this communication between nodes and the local CPU-GPU costs as there is an inherent *double-copy* when using MPI library calls for a set of one or more CPU/GPU systems [105].

Establishing a direct and dynamic *relation* among all the defined software, hardware, and algorithmic factors is necessary to elicit optimal performance boost for the presented candidate application. This same judicious application of software and algorithmic methodologies are needed for many other HPC computational modeling applications as the iterative solution to the sparse matrix system defined by the presented candidate application is common to many scientific and engineering applications [20, 21, 23, 39, 55, 80] that wish to fully utilize the substantial performance boosting capabilities of not only GPU accelerators but the inexorable domination of multi-core CPUs [3, 4, 114].

CHAPTER 6

Summary and Future Directions

The major conclusions of this dissertation can be summarized as follows:

- (i) The relationship of software and hardware factors on the performance of computationally intense applications that wish to execute within the context of the modern CPU/GPU computing systems must be judiciously applied for optimal performance.
- (ii) A predictive performance model was adapted for this research and is within the range of acceptable normalized error for functionality. This model can be used to assist with the proper determination of cost/benefit optimal manipulation of software and hardware factors.
- (iii) Intra-nodal communication and local CPU/GPU host communication can be deleterious to performance benefits for multiple CPU/GPU computing environments and the asymptotic upper bound on this communicational cost was calculated as asymptotically bound to cubic values with *data locality*.
- (iv) The more regular an input matrix being solved by a CPU/GPU computing system, single or multiple node, the more exposed software factors are to final performance whereas the less regular a resulting matrix system, the greater the impact of hardware.

Computing systems are fast approaching a time when the non-deterministic paradigm of parallelism inherent in multi-cored architectures like the GPU will become common-place. High Performance Computing applications wishing to harness this computational power optimally will have to be adjusted as per three categories of factors – software, hardware, and algorithmic. Computing system environments will continue to evolve but the basic understanding of these performance factors will provide solid foundations upon which robust and efficient legacy and new computational modeling applications can be developed.

Chapters 2 and 3 provided the underlying hardware architectural and software algorithmic principles of two separate CPU/GPU computing systems defined in this work as **System A** and **System B**. Algorithmic factor adjustments such as switching from a one thread per row to one *warp* per row to solve the sparse matrix system $\underline{Ax} = \underline{b}$ engender an immediate performance boost. The same statement can be made for software factor adjustments such as data structure layout via the CSR to BCSR2x2, as the general improvement in locality mitigated the lack of real memory cache inherent to GPU devices. **System B** illustrated a distinct hardware

architectural advantage over **System A**, providing more than 3-times processing cores as well as 4-times the number of registers and memory devices that executed on both sides of the clock pulse – affecting a *double-pumped* graphics pipeline. These initial chapter results were reflected for both the single and multiple CPU/GPU computing systems in chapters 4 and 5, with the added complexity of MPI communication for the latter.

Chapter 4 also produced a computational complexity analysis of the CPU/GPU computing system that was used to project the performance of the presented candidate application within the context of both **System A** and **System B** machine environments. Adjusting the software and hardware variables in the complexity equation reflected actual performance results to within reasonable limits corroborating the interdependence of software and hardware factors of the CPU/GPU computing architecture at a mathematical level. The introduction of *multiple* CPU/GPU computing systems in chapter 5 further advanced the concept of these performance factors as the mathematical complexity was shown to be an exponential factor of the number of SMPs per system utilized. Chapter 5 also exposed the cost of *intra-nodal* and *CPU-GPU local host* communication as a correlation of the percentage of locally defined non-zero elements held by a given GPU *device registers* and the *factors off* from the calculated *ideal parallelism* via domain decomposition as multiple processors/nodes – found as a negative factor on performance that is *cubic* in nature. The performance results determined with the presented candidate application can be applied to other computationally intensive HPC applications as well. Chapter 5 also revealed that the less regular input mesh defined by **10FT** the less effect locality plays with regards to data compression formats – due to smaller likelihood of dense sub-matrices that are critical to blocked compression formats e.g., BCSR2x2.

The presented candidate application is designed around computational elements built up using the FEM methodology, resulting in a sparse matrix system that is a well-documented point of computational intensity [21, 48, 71, 79, 110, 112]. The solution of systems involving sparse matrices is a common paradigm in the HPC modeling applications, all facing the same computational dilemma – how to optimally solve these algorithms using modern computing environments. Thus, the methodologies presented in this work can be applied to a wide range of computationally intensive applications built around sparse matrix systems and their solution in the computational modeling analysis.

The current popularity of the GPU as a computationally powerful co-processor will continue to grow as demand for more powerful machines to execute HPC applications grows – this will be exacerbated by the trend in mobile computing. The on-chip architectures of mobile computing tables and smart phones have provided a new and interesting opportunity for GPGPU computing – fused addresses [115]. The PCIe CPU-GPU communication bottleneck is well documented [21, 48, 71, 79, 110, 112] but a fusing of CPU and GPU on the same chip will likely

change this but will also create some new issues, e.g. memory device I/O. The fused systems, such as AMD APC processor use the slower DDR memory device rather than GDDR of the GPU resulting in the unusual situation of an efficient sparse matrix solution but with the opposite effect on dense matrix systems [114, 115].

The inexorable growth in multi-cored CPUs will also provide more computationally intensive power and a unique dynamic will develop as the GPU gets closer to the flexible memory structure of the CPU, and vice versa such as processors like Intel's Sandy Bridge [114]. The developer wishing to attain optimal performance with these new machines will need to understand the intricacies of the software, hardware, and algorithmic factors as presented in this work.

References

- 1 Boggan, S.K., and Pressel, D.M.: 'GPUs: An Emerging Platform for General-Purpose Computation', in Editor (Ed.)^(Eds.): 'Book GPUs: An Emerging Platform for General-Purpose Computation' (Army Research Lab, 2007, edn.), pp. 50
- 2 Fatahalian, K., and Houston, M.: 'GPUs: A Closer Look', ACM Queue, 2008, 6, (2), pp. 10
- 3 Ross, P.E.: 'Why CPU Frequency Stalled', IEEE Spectrum, 2008, 45, (4), pp. 1
- 4 Tian, D.Z.: 'Editorial (Moore's Law)', IEEE Potentials, 2008, 27, (6), pp. 3
- 5 Kang, S., Choi, H.J., Kim, C.H., Chung, S.W., Kwon, D., and Na, J.C.: 'Exploration of CPU/GPU co-execution: from the perspective of performance, energy, and temperature', in Editor (Ed.)^(Eds.): 'Book Exploration of CPU/GPU co-execution: from the perspective of performance, energy, and temperature' (ACM New York, NY, USA 2011, edn.), pp. 38-43
- 6 Hong, S., and Kim, H.: 'An integrated GPU power and performance model', in Editor (Ed.)^(Eds.): 'Book An integrated GPU power and performance model' (ACM New York, NY, USA, 2010, edn.), pp. 280-289
- 7 Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P.: 'Brook for GPUs: Stream Computing on Graphics Hardware', ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2004 2004, 23, (3), pp. 777-786
- 8 Rumpf, M., and Strzodka, R.: 'Graphics Processor Units: New Prospects for Parallel Computing', in Bruaset, A.M.a.T., Aslak (Ed.): 'Numerical Solution of Partial Differential Equations on Parallel Computers' (Springer, 2005), pp. 89-134
- 9 Silberschatz, A., Galvin, P., and Gagne, G.: 'Applied Operating System Concepts: windows XP update' (John Wiley & Sons, Inc., 2003, 1st edn. 2003)
- 10 Silberschatz, A., Galvin, P., and Gagne, G.: 'Applied Operating System Concepts' (John Wiley & Sons, Inc., 2000, 1 edn. 2000)
- 11 Ross, P.E.: 'Why CPU Frequency Stalled', IEEE Spectrum, 2008, 45, (4), pp. 72-72
- 12 Patterson, D.: 'Computer Organization and Design : The Hardware/Software Interface' (Elsevier Science Ltd, 1997. 1997)
- 13 Patterson, D.A., and Hennessy, J.L.: 'Computer Organization & Design: The Hardware/Software Interface' (Morgan Kaufmann Publishers, Inc., 1998, 2nd edn. 1998)
- 14 Liang, Y.: 'The Use of Parallel Polynomial Preconditioners in the Solution of Systems of Linear Equations', University of Ulster, 2005
- 15 Cheng, H.: 'Vector Pipelining, Chaining, and Speed on the IBM 3090 and Cray X-MP', Computer, 1989, 22, (9), pp. 9
- 16 Sivasubramaniam, A., Singla, A., Ramachandran, U., and Venkateswaran, H.: 'Machine Abstractions and Locality Issues in Studying Parallel Systems', in Editor (Ed.)^(Eds.): 'Book Machine Abstractions and Locality Issues in Studying Parallel Systems' (Georgia Institute of Technology, 1993, edn.), pp.

- 17 Thomas, S.: 'Preconditioned Conjugate Gradient Methods for Semiconductor Device Simulation on a CRAY C90 Vector Processor'. Proc. VECPAR '96 Selected papers from the Second International Conference on Vector and Parallel Processing, 1997 1996 pp. Pages
- 18 Thomas, S.: 'Preconditioned conjugate gradient methods for semiconductor device simulation on a CRAY C90 vector processor', Vector and Parallel Processing — VECPAR'96, 1997, 1215
- 19 Tagaya, S., Nishida, M., Hagiwara, T., Yanagawa, T., Yokoya, Y., Takahara, H., stadler, J.o., and Galle, M.: 'The NEC SX-8 Vector Supercomputer System': 'High Performance Computing on Vector Systems' (Springer-Verlag Berlin, Heidelberg, 2006), pp. Part-1, 3-24
- 20 Bustamam, A., Burrage, K., and Hamilton, N.A.: 'Fast Parallel Markov Clustering in Bioinformatics Using Massively Parallel Computing on GPU with CUDA and ELLPACK-R Sparse Format', IEEE/ACM Trans Comput Biol Bioinform., 2012, 3, (9), pp. 13
- 21 Corrigan, A., Camelli, F.F., Lohner, R., and Wallin, J.: 'Running unstructured grid-based CFD solvers on modern graphics hardware', International Journal for Numerical Methods in Fluids, 2011, 66, (2), pp. 221-229
- 22 Grozea, C., Bankovic, Z., and Laskov, P.: 'FPGA vs. Multi-Core CPUs vs. GPUs: Hands-on Experience with a Sorting Application': 'Facing the multicore-challenge ' (Springer-Verlag Berlin, Heidelberg 2010), pp. 105-117
- 23 Hamada, T., Narumi, T., Yasuoka, K., Nitadori, K., and Taiji, M.: '42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence', in Editor (Ed.)^(Eds.): 'Book 42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence' (ACM New York, NY, USA, 2009, edn.), pp. 12
- 24 Luebke, D., and Humphreys, G.: 'How GPUs Work', IEEE Computer, 2007, 2007
- 25 Luebke, D.: 'CUDA: Scalable parallel programming for high-performance scientific computing', in Editor (Ed.)^(Eds.): 'Book CUDA: Scalable parallel programming for high-performance scientific computing' (ACM New York, NY, USA 2008, edn.), pp. 836 - 838
- 26 El-Ghazawi, T.A., Cantonnet, F., Yao, Y., Annareddy, S., and Mohamed, A.S.: 'Benchmarking parallel compilers: A UPC case study', Future Generation Computer Systems - Systems performance analysis and evaluation 2006, 22, (7), pp. 11
- 27 Polze, A., and Troger, P.: 'Trends and challenges in operating systems—from parallel computing to cloud computing', Concurrency and Computation: Practice & Experience, 2012, 24, (7), pp. 10

- 28 Wilkinson, B., and Allen, M.: 'Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers' (Pearson Education, Inc., 2005, 2nd edn. 2005)
- 29 Haney, R.H.: 'Study and Evaluation of Domain Decomposition Approaches in two Parallel Software Code Developments for Process Flow Modeling in Liquid Composite Molding', North Carolina A & T State University, 2006
- 30 Mohan, D.R., Shires, D., and Mark, A.: 'Scalable Large Scale Process Modeling and Simulations in Liquid Composite Molding': 'Computational Science - ICCS 2001' (Springer Berlin Heidelberg, 2001), pp. 1199-1208
- 31 Angel, E., and Shreiner, D.: 'An introduction to shader-based OpenGL programming', in Editor (Ed.)^(Eds.): 'Book An introduction to shader-based OpenGL programming' (ACM, 2009, edn.), pp.
- 32 Udupa, A., Govindarajan, R., and Thazhuthaveetil, M.J.: 'Software Pipelined Execution of Stream Programs on GPUs', in Editor (Ed.)^(Eds.): 'Book Software Pipelined Execution of Stream Programs on GPUs' (IEEE Computer Society Washington, DC, USA, 2009, edn.), pp. 200-209
- 33 Chen, J.X., and Wegman, E.J.: 'Foundations of 3D Graphics Programming: Using JOGL and Java3D' (Springer-Verlag London Limited, 2006, 1 edn. 2006)
- 34 Hearn, D., and Baker, P.M.: 'Computer Graphics C version - 2nd Edition' (Pearson Education, 1997, 1996, 2nd edn. 1996)
- 35 <http://www.opengl-tutorial.org>, accessed 12-02-2012 2012
- 36 Leeser, M., Yablonski, D., Brooks, D., and King, L.S.: 'The Challenges of Writing Portable, Correct and High Performance Libraries for GPUs', ACM SIGARCH Computer Architecture News, 2011, 39, (4), pp. 5
- 37 Nvidia: 'CUDA C BEST PRACTICES GUIDE', in Editor (Ed.)^(Eds.): 'Book CUDA C BEST PRACTICES GUIDE' (Nvidia Corporation, 2011, edn.), pp. 76
- 38 Nvidia: 'PARALLEL THREAD EXECUTION ISA VERSION 3.1', in Editor (Ed.)^(Eds.): 'Book PARALLEL THREAD EXECUTION ISA VERSION 3.1' (Nvidia Corporation, 2012, edn.), pp. 241
- 39 Ayanda, D., and Adejumo, Y.: 'A Prototype Model of High Performance Computing Using Beowulf Cluster', International Journal of Emerging Sciences, 2011, 1, (4), pp. 696-705
- 40 Goddeke, D., Wobker, H., Strzodka, R., Mohd-Yusof, J., McCormick, P., and Turek, S.: 'Co-Processor Acceleration of an Unmodified Parallel Solid Mechanics Code with FeastGPU', International Journal of Computational Science and Engineering, 2009, 4, (4), pp. 254-269
- 41 Lu, F., Song, J., Yin, F., and Zhu, X.: 'Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters', Computer Physics Communications, 2011, 183, (6), pp. 1172-1181

- 42 Mohan, R., Ngo, N.D., Tamma, K.K., and Fickie, K.D.: 'Three-Dimensional Resin Transfer Molding Process: Developments for Thick Composite Manufacturing Applications', in Editor (Ed.)^(Eds.): 'Book Three-Dimensional Resin Transfer Molding Process: Developments for Thick Composite Manufacturing Applications' (U.S. Army Research Laboratory, 1996, edn.), pp. 32
- 43 Mohan, R.V., Ngo, N.D., Tamma, K.K., and Fickie, K.D.: 'On a Pure Finite-Element-Based Methodology for Resin Transfer Mold Filling Simulations', in Editor (Ed.)^(Eds.): 'Book On a Pure Finite-Element-Based Methodology for Resin Transfer Mold Filling Simulations' (Army Research Lab, 1996, edn.), pp. 18
- 44 Chandrupatla, T.R., and Belegundu, A.D.: 'Introduction To Finite Elements In Engineering' (Prentice-Hall, Inc., 2002, 3 edn. 2002)
- 45 Rao, S.S.: 'Applied Numerical Methods For Engineers And Scientists' (Prentice-Hall, Inc., 2002, 1st edn. 2002)
- 46 Shewchuk, J.R.: 'An Introduction to the Conjugate Gradient Method Without the Agonizing Pain', in Editor (Ed.)^(Eds.): 'Book An Introduction to the Conjugate Gradient Method Without the Agonizing Pain' (Carnegie Mellon University, 1994, edn.), pp. 64
- 47 Mohan, D.R., Ngo, N.D., and Tamma, K.K.: 'On a pure finite-element-based methodology for resin transfer molds filling simulations', Polymer Engineering & Science, 1999, 39, (1), pp. 26-43
- 48 Baskaran, M.M., and Bordawekar, R.: 'Optimizing Sparse Matrix-Vector Multiplication on GPUs', in Editor (Ed.)^(Eds.): 'Book Optimizing Sparse Matrix-Vector Multiplication on GPUs' (IBM Research, 2008, edn.), pp. 11
- 49 Bell, N., and Garland, M.: 'Efficient Sparse Matrix-Vector Multiplication on CUDA', in Editor (Ed.)^(Eds.): 'Book Efficient Sparse Matrix-Vector Multiplication on CUDA' (NVIDIA Corporation, 2008, edn.), pp. 32
- 50 Buatois, L., Caumon, G., and Levy, B.: 'Concurrent number cruncher: a GPU implementation of a general sparse linear solver', International Journal of Parallel, Emergent and Distributed Systems, 2009, 24, (3), pp. 18
- 51 Helfenstein, R., and Koko, J.: 'Parallel preconditioned conjugate gradient algorithm on GPU', Journal of Computational and Applied Mathematics, 2011, 236, (15), pp. 6
- 52 Shahnaz, R., Usman, A., and Chughtai, I.R.: 'Review of Storage Techniques for Sparse Matrices', in Editor (Ed.)^(Eds.): 'Book Review of Storage Techniques for Sparse Matrices' (IEEE, 2005, edn.), pp. 1-7
- 53 Hugues, M.R., and Petiton, S.G.: 'Sparse Matrix Formats Evaluation and Optimization on a GPU', in Editor (Ed.)^(Eds.): 'Book Sparse Matrix Formats Evaluation and Optimization on a GPU' (IEEE Computer Society Washington, DC, USA, 2010, edn.), pp. 122-129

- 54 Rehman, M.S.: 'Exploring Irregular Memory Access Applications on the GPU', International Institute of Information Technology, 2010
- 55 De Jong, M.A.: 'Developing a CUDA solver for large sparse matrices for MARIN'. Master thesis, Delft University of Technology, 2012
- 56 Corporation, A.: 'AMD Family 10h Server and Workstation Processor Power and Thermal Data Sheet', in Editor (Ed.)^(Eds.): 'Book AMD Family 10h Server and Workstation Processor Power and Thermal Data Sheet' (2010, edn.), pp. 98
- 57 Corporation, I.: 'Intel® Xeon® Processor 5600 Series: The Next Generation of Intelligent Server Processors', in Editor (Ed.)^(Eds.): 'Book Intel® Xeon® Processor 5600 Series: The Next Generation of Intelligent Server Processors' (Intel Corporation, 2010, edn.), pp. 8
- 58 Corporation, N.: 'NVIDIA Quadro® FX 5600 Datasheet', in Editor (Ed.)^(Eds.): 'Book NVIDIA Quadro® FX 5600 Datasheet' (2008, edn.), pp. 2
- 59 Corporation, N.: 'TESLA M2050 AND TESLA M2070/M2070Q DUAL-SLOT COMPUTING PROCESSOR MODULES', in Editor (Ed.)^(Eds.): 'Book TESLA M2050 AND TESLA M2070/M2070Q DUAL-SLOT COMPUTING PROCESSOR MODULES' (Nvidia Corporation, 2010, edn.), pp. 18
- 60 Nvidia: 'NVIDIA GeForce 8800 Architecture Technical Brief', in Editor (Ed.)^(Eds.): 'Book NVIDIA GeForce 8800 Architecture Technical Brief' (Nvidia Corporation, 2006, edn.), pp. 55
- 61 Nvidia: 'NVIDIA's Next Generation CUDA Compute Architecture: Fermi - Whitepaper', in Editor (Ed.)^(Eds.): 'Book NVIDIA's Next Generation CUDA Compute Architecture: Fermi - Whitepaper' (NVIDIA Corporation, 2009, edn.), pp. 22
- 62 Nvidia: 'NVIDIA CUDA C Programming Guide: Version 4.2', in Editor (Ed.)^(Eds.): 'Book NVIDIA CUDA C Programming Guide: Version 4.2' (Nvidia Corporation, 2012, edn.), pp. 173
- 63 Komatitsch, D., Michea, D., and Erlebacher, G.: 'Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA', Journal of Parallel Computing, 2009, 69, (5), pp. 9
- 64 Kuznik, F., Obrecht, C., Rusaouen, G., and Roux, J.-J.: 'LBM based flow simulation using GPU computing processor', Computers & Mathematics with Applications, 2010, 59, (7), pp. 12
- 65 Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V.: 'Parallel Computing Experiences with CUDA ', Micro, IEEE 2008, 28, (4), pp. 13-27
- 66 Parakh, A.: 'Performance estimation and application mapping on different GPUs'. Proc. HiPC - High Performance Computing Confrence, Pune, INDIA, December 18-21, 2012 2012 pp. Pages

- 67 Bernaschi, M., Bisson, M., and Rossetti, D.: 'Benchmarking of communication techniques for GPUs', *Journal of Parallel and Distributed Computing*, 2013, 73, (2), pp. 5
- 68 Micikevicius, P.: '3D Finite Difference Computation on GPUs using CUDA', in Editor (Ed.)^(Eds.): 'Book 3D Finite Difference Computation on GPUs using CUDA' (ACM New York, NY, USA, 2009, edn.), pp.
- 69 Papadrakakis, M., Stavroulakis, G., and Karatarakis, A.: 'A new era in scientific computing: Domain decomposition methods in hybrid CPU-GPU architectures', *Computer Methods in Applied Mechanics and Engineering*, 2011, 200, (13-16), pp. 1490-1508
- 70 Di, P., Wu, H., Xue, J., Wang, F., and Yang, C.: 'Parallelizing SOR for GPGPUs using alternate loop tiling', *Journal of Parallel Computing*, 2012, 38, (6-7), pp. 18
- 71 Goddeke, D., Strzodka, R., and Turek, S.: 'Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations', *International Journal of Parallel, Emergent and Distributed Systems* 2007, 22, (4), pp. 221-256
- 72 Kothapalli, K., Mukherjee, R., Rehman, M.S., Patidar, S., Narayanan, P.J., and Srinathan, K.: 'A Performance Prediction Model for the CUDA GPGPU Platform', in Editor (Ed.)^(Eds.): 'Book A Performance Prediction Model for the CUDA GPGPU Platform' (IEEE, 2009, edn.), pp. 463 - 472
- 73 Lukash, M., and Rupp, K.: 'Sparse Approximate Inverse Preconditioners for Iterative Solvers on GPUs', in Editor (Ed.)^(Eds.): 'Book Sparse Approximate Inverse Preconditioners for Iterative Solvers on GPUs' (Society for Computer Simulation International 2012, edn.), pp.
- 74 Huo, X., Ravi, V., Ma, W., and Agrawal, G.: 'An Execution Strategy and Optimized Runtime Support for Parallelizing Irregular Reductions on Modern GPUs', in Editor (Ed.)^(Eds.): 'Book An Execution Strategy and Optimized Runtime Support for Parallelizing Irregular Reductions on Modern GPUs' (ACM New York, NY, USA, 2011, edn.), pp. 2-11
- 75 Wald, I.: 'Active thread compaction for GPU path tracing', in Editor (Ed.)^(Eds.): 'Book Active thread compaction for GPU path tracing' (ACM New York, NY, USA, 2011, edn.), pp. 51-58
- 76 Hewlett-Packard Development Company, L.P.: 'QuickSpecs NVIDIA Quadro FX 5600 PCIe Graphics Card', in Editor (Ed.)^(Eds.): 'Book QuickSpecs NVIDIA Quadro FX 5600 PCIe Graphics Card' (2009, edn.), pp. 4
- 77 Gou, C., and Gaydadjiev, G.N.: 'Elastic Pipeline: Addressing GPU On-chip Shared Memory Bank Conflicts', in Editor (Ed.)^(Eds.): 'Book Elastic Pipeline: Addressing GPU On-chip Shared Memory Bank Conflicts' (ACM New York, NY, USA, 2011, edn.), pp.

- 78 Nvidia: 'CUDA CUBLAS Library: Version 1.1', in Editor (Ed.)^(Eds.): 'Book CUDA CUBLAS Library: Version 1.1' (Nvidia Corporation, 2007, edn.), pp. 84
- 79 Corporation, N.: 'CUDA CUSPARSE Users Guide', in Editor (Ed.)^(Eds.): 'Book CUDA CUSPARSE Users Guide' (Nvidia Corporation, 2012, v5.0 edn.), pp. 123
- 80 Bahi, J.M., Couturier, R., and Khodja, L.Z.: 'Parallel GMRES implementation for solving sparse linear systems on GPU clusters', in Editor (Ed.)^(Eds.): 'Book Parallel GMRES implementation for solving sparse linear systems on GPU clusters' (Society for Computer Simulation International San Diego, CA, USA, 2011, edn.), pp. 12-19
- 81 Kruger, J., and Westermann, R.: 'Linear Algebra Operators for GPU Implementation of Numerical Algorithms', ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2003, 2003, 22, (3), pp. 8
- 82 Holk, E., Byrd, W., Mahajan, N., Wilcock, J., Chauhan, A., and Lumsdaine, A.: 'Declarative Parallel Programming for GPUs': 'Advances in Parallel Computing, Volume 22: Applications, Tools and Techniques on the Road to Exascale Computing' (IOS Press, 2011)
- 83 Kahan, W.: 'IEEE Standard 754 for Binary Floating-Point Arithmetic', in Editor (Ed.)^(Eds.): 'Book IEEE Standard 754 for Binary Floating-Point Arithmetic' (University of California Berkeley CA, USA, 1997, edn.), pp. 30
- 84 Dimitrov, M., Mantor, M., and Zhou, H.: 'Understanding Software Approaches for GPGPU Reliability', in Editor (Ed.)^(Eds.): 'Book Understanding Software Approaches for GPGPU Reliability' (ACM New York, NY, USA, 2009, edn.), pp. 94-104
- 85 Hillesland, K., and Lastra, A.: 'GPU floating-point paranoia'. Proc. ACM Workshop on General Purpose Computing on Graphics Processors In ACM Workshop on General Purpose Computing on Graphics Processors 2004 2004 pp. Pages
- 86 Whitehead, N., and Fit-Florea, A.: 'Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs', in Editor (Ed.)^(Eds.): 'Book Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs' (Nvidia, 2011, edn.), pp. 7
- 87 Castaldo, A.M.: 'ERROR ANALYSIS OF VARIOUS FORMS OF FLOATING POINT DOT PRODUCTS', The University of Texas at San Antonio, 2007
- 88 Volkov, V., and Demmel, J.W.: 'Benchmarking GPUs to tune dense linear algebra'. Proc. International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008., Austin, TX, USA, 15-21 Nov. 2008 2008 pp. Pages
- 89 Mohan, R.V., Ngo, N.D., and Tamma, K.K.: 'On a Pure Finite Element Methodology for Resin Transfer Mold Filling Simulations', Polymer Engineering and Science, 1999, 39, pp. 26-43
- 90 Fatahalian, K., Sugerman, J., and Hanrahan, P.: 'Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication', in Editor (Ed.)^(Eds.): 'Book

- Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication' (ACM New York, NY, USA 2004, edn.), pp. 133-137
- 91 Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C.: 'Introduction To Algorithms' (The MIT Press, 2001, 2nd edn. 2001)
 - 92 Resios, A.: 'GPU performance prediction using parametrized models'. Masters, Utrecht University, 2011
 - 93 Zhang, Y., and Owens, J.D.: 'A quantitative performance analysis model for GPU architectures'. Proc. High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on San Antonio, TX, 12-16 Feb. 2011 2011 pp. Pages
 - 94 Wang, H., Potluri, S., Luo, M., Singh, A.K., Sur, S., and Panda, D.K.: 'MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters', Computer Science - Research and Development, 2011, 26, (3-4), pp. 257-266
 - 95 Karunadasa, N.P., and Ranasinghe, D.N.: 'Accelerating High Performance Applications with CUDA and MPI'. Proc. 2009 International Conference on Industrial and Information Systems (ICIIS), Sri Lanka, December 28-31, 2009 2009 pp. Pages
 - 96 Tipparaju, V., and Vetter, J.S.: 'GA-GPU: Extending a Library-based Global Address Space Programming Model for Scalable Heterogeneous Computing Systems', in Editor (Ed.)^(Eds.): 'Book GA-GPU: Extending a Library-based Global Address Space Programming Model for Scalable Heterogeneous Computing Systems' (ACM New York, NY, USA 2012, edn.), pp. 53-64
 - 97 Wang, L., Huang, M., Narayana, V., and El-Ghazawi, T.A.: 'Scaling Scientific Applications on Clusters of Hybrid Multicore/GPU Nodes', in Editor (Ed.)^(Eds.): 'Book Scaling Scientific Applications on Clusters of Hybrid Multicore/GPU Nodes' (ACM New York, NY, USA, 2011, edn.), pp.
 - 98 Wilkinson, B., and Allen, M.: 'Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers' (Pearson Prentice Hall, 2005, 2 edn. 2005)
 - 99 Sun Microsystems, I.: 'Sun HPC ClusterTools 8.2 (software tools)', in Editor (Ed.)^(Eds.): 'Book Sun HPC ClusterTools 8.2 (software tools)' (Oracle, Inc., 2009, edn.), pp.
 - 100 Geist, G.A., Kohl, J.A., and Papadopoulos, P.M.: 'PVM and MPI: A comparison of features', Calculateurs Paralleles, 1996, 8, (2)
 - 101 Song, J.P., and Shires, D.: 'Central Processing Unit/Graphics Processing Unit (CPU/GPU) Hybrid Computing of Synthetic Aperture Radar Algorithm', in Editor (Ed.)^(Eds.): 'Book Central Processing Unit/Graphics Processing Unit (CPU/GPU) Hybrid Computing of Synthetic Aperture Radar Algorithm' (U.S. Army Research Laboratory, 2010, edn.), pp.

- 102 Khajeh-Saeed, A., and Perot, J.B.: 'Computational Fluid Dynamics Simulations Using Many Graphics Processors', *Computing in Science & Engineering*, 2011, 14, (3), pp. 10-19
- 103 Komatitsch, D., Michea, D., and Erlebacher, G.: 'Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA', *Journal of Parallel and Distributed Computing*, 2009, 69, (5), pp. 451-460
- 104 Fengshun, L., Song, J., Yin, F., and Zhu, X.: 'Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters', *Computer Physics Communications*, 2012, 183, (6), pp. 1172-1181
- 105 Ji, F., Ajiy, A.M., Dinanz, J., Buntinasz, D., Balajiz, P., Fengy, W.-c., and Ma, X.: 'Efficient Intranode Communication in GPU-Accelerated Systems'. *Proc. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012 IEEE 26th International Shanghai 21-25 May 2012 2012 pp. Pages
- 106 Belviranli, M.E., Bhuyan, L.N., and Gupta, R.: 'A Dynamic Self-Scheduling Scheme for Heterogeneous Multiprocessor Architectures', *ACM Transactions on Architecture and Code Optimization (TACO) - Special Issue on High-Performance Embedded Architectures and Compilers*, 2013, 9, (4), pp. 19
- 107 Filipovic, J., Peterlik, I., and Fousek, J.: 'GPU Acceleration of Equations Assembly in Finite Elements Method – Preliminary Results', *SAAHPC: Symposium on Application Accelerators in HPC*, 2009
- 108 Fousek, J., Filipovic, J., and Madzin, M.: 'Automatic Fusions of CUDA-GPU Kernels for Parallel Map', *ACM SIGARCH Computer Architecture News*, 2011, 39, (4), pp. 1
- 109 Khajeh-Saeed, A., and Perot, J.B.: 'Computational Fluid Dynamics Simulations Using Many Graphics Processors', *Computing in Science & Engineering*, 2012, 14, (3), pp. 9
- 110 Goddeke, D., Strzodka, R., Mohd-Yusof, J., McCormick, P., Buijssen, S.H.M., Grajewski, M., and Turek, S.: 'Exploring weak scalability for FEM calculations on a GPU-enhanced cluster', *Journal of Parallel Computing*, 2007, 33, (10-11), pp. 685-699
- 111 Samuels, M.L., and Witmer, J.A.: 'Statistics for the Life Sciences' (Pearson Education, Inc., 2003, 3 edn. 2003)
- 112 Che, S., Sheaffer, J.W., and Skadron, K.: 'Dymaxion: Optimizing Memory Access Patterns for Heterogeneous Systems', in Editor (Ed.)^(Eds.): 'Book Dymaxion: Optimizing Memory Access Patterns for Heterogeneous Systems' (ACM New York, NY, USA, 2011, edn.), pp.
- 113 Steinberger, M., Kainz, B., Kerbl, B., Hauswiesner, S., Kenzel, M., and Schmalstieg, D.: 'Softshell: Dynamic Scheduling on GPUs', *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2012*, 2012, 31, (6), pp. 12
- 114 Solutions, A.E.: 'Intel Sandy Bridge Brings Many Benefits the PC/104 Form Factor', in Editor (Ed.)^(Eds.): 'Book Intel Sandy Bridge Brings Many Benefits the PC/104 Form Factor' (Embedded Solutions, 2011, edn.), pp. 5

- 115 Spafford, K.L., Meredith, J.S., Lee, S., Li, D., Roth, P.C., and Vetter, J.S.: 'The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Computing Architectures', in Editor (Ed.)^(Eds.): 'Book The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Computing Architectures' (ACM New York, NY, USA, 2012, edn.), pp. 103-112

Appendix A

The CUDA Kernel code and associated functions and structures for the execution of sparse matrix-vector multiplication discussed in chapter 3 are presented below.

CUDA File (matvec.cu):

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cuda.h>
#include <cublas.h>

#define BLOCK_SIZE 16

// HOST-Side C-Code interface for generalized matrix multiplication operation of the form
// 'Ax = b' using CRS compression.
extern "C" void mul_multBlocks_CRS(float *val, unsigned int vLength, unsigned int *rp,
    unsigned int rpLength, unsigned int *cp, unsigned int cpLength, float *b,
    float *x, unsigned int m, unsigned int n, unsigned int p, float &time);

// HOST-Side C-Code interface for generalized matrix multiplication operation of the form
// 'Ax = b' using BCRS 2x2 compression.
extern "C" void mul_multBlocks_BCRS(float *val, unsigned int vLength, unsigned int *rp,
    unsigned int rpLength, unsigned int *cp, unsigned int cpLength, float *b,
    float *x, unsigned int m, unsigned int n, unsigned int p, float &time);

// GPU-Code for generalized matrix multiplication operation of the form 'Ax = b' for CSR
// format.
__global__ void mul_multipleblocks_CRS(float *val, float *b, float *x, uint2 *rp,
    unsigned int *cp, unsigned int m, unsigned int n);

// GPU-Code for generalized matrix multiplication operation of the form 'Ax = b' for BCRS 2x2
// format.
__global__ void mul_multipleblocks_BCRS(float4 *val, float2 *b, float2 *x, uint2 *rp,
    unsigned int *cp, unsigned int m, unsigned int n);

// Computes the current THREAD index.
__device__ unsigned int compute_thread_index() {
    return (blockIdx.x*BLOCK_SIZE*BLOCK_SIZE
```

```

        + blockIdx.y*BLOCK_SIZE*BLOCK_SIZE*gridDim.x
        + threadIdx.x + threadIdx.y*BLOCK_SIZE);
    }

void mul_multBlocks_CRS(float *val, unsigned int vLength, unsigned int *rp,
    unsigned int rpLength, unsigned int *cp, unsigned int cpLength, float *b, float *x,
    unsigned int m, unsigned int n, unsigned int p, float &time) {
    // Timing this operation.
    cudaEvent_t start, stop; time = 0.0f;

    // Initialize EVENT Timers - CUDA.
    cudaEventCreate(&start); cudaEventCreate(&stop);

    // Variables to be placed on GPU.
    float *val_d = NULL; float *b_d = NULL;
    float *x_d = NULL; uint2 *rp_d = NULL;
    unsigned int *cp_d = NULL;

    // Compute ROW "pointer" BOUNDS to be pushed on the GPU
    uint2 *cpu_rp = new uint2[rpLength - 1];
    {
        for(unsigned int i = 0; i < rpLength - 1; i++)
        {
            cpu_rp[i].x = rp[i];
            cpu_rp[i].y = rp[i + 1];
        }
    }

    // Allocate and initialize values for GPU
    cudaMalloc((void**)&val_d, vLength*sizeof(float));
    cudaMalloc((void**)&x_d, m*p*sizeof(float));
    cudaMalloc((void**)&b_d, n*p*sizeof(float));
    cudaMalloc((void**)&cp_d, cpLength*sizeof(unsigned int));
    cudaMalloc((void**)&rp_d, rpLength*sizeof(uint2));

    cudaMemcpy(cp_d, cp, cpLength*sizeof(unsigned int), cudaMemcpyHostToDevice);
    cudaMemcpy(rp_d, cpu_rp, rpLength*sizeof(uint2), cudaMemcpyHostToDevice);
    cudaMemcpy(val_d, val, vLength*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(b_d, b, n*p*sizeof(float), cudaMemcpyHostToDevice);

```

```

// Calculate dimensions for GPU Device
dim3 grid; dim3 block;
grid.x = (unsigned int)(sqrt((float)n)/BLOCK_SIZE + 1);
grid.y = (unsigned int)(sqrt((float)n)/BLOCK_SIZE + 1);
block.x = BLOCK_SIZE; block.y = BLOCK_SIZE;

cudaEventRecord(start, 0);

// Kernel call
mul_multipleblocks_CRS<<<grid, block>>>(val_d, b_d, x_d, rp_d, cp_d, m, n);

// "Record" the stopping of this EVENT - i.e. return from kernel call.
cudaEventRecord(stop, 0); cudaEventSynchronize(stop);

// Get the amount of time elapsed (in milliseconds) and DESTROY the CUDA timer
// objects.
cudaEventElapsedTime(&time, start, stop);
cudaEventDestroy(start); cudaEventDestroy(stop);

// Retrieve results pointed by 'x_d'
cudaMemcpy(x, x_d, m*p*sizeof(float), cudaMemcpyDeviceToHost);

// Free Memory - CPU.
delete [] cpu_rp;

// Free Memory - GPU.
cudaFree(val_d);
cudaFree(b_d);
cudaFree(x_d);
cudaFree(rp_d);
cudaFree(cp_d);
}

void mul_multBlocks_BCRS(float *val, unsigned int vLength, unsigned int *rp,
    unsigned int rpLength, unsigned int *cp, unsigned int cpLength, float *b, float *x,
    unsigned int m, unsigned int n, unsigned int p, float &time) {
    // Timing this operation.

```

```

cudaEvent_t start, stop; time = 0.0f;

// Initialize EVENT Timers - CUDA.
cudaEventCreate(&start); cudaEventCreate(&stop);

// Variables to be placed on GPU.
float4 *val_d = NULL; float2 *x_d = NULL;
float2 *b_d = NULL; uint2 *rp_d = NULL;
unsigned int *cp_d = NULL;

// Compute ROW "pointer" BOUNDS to be pushed on the GPU.
uint2 *cpu_rp = new uint2[rpLength - 1];
{
    for(unsigned int i = 0; i < rpLength - 1; i++)
    {
        cpu_rp[i].x = rp[i];
        cpu_rp[i].y = rp[i + 1];
    }
}

// Allocate and initialize values for GPU.
cudaMalloc((void**)&val_d, vLength*sizeof(float4));
cudaMalloc((void**)&x_d, m*p*sizeof(float2));
cudaMalloc((void**)&b_d, n*p*sizeof(float2));
cudaMalloc((void**)&cp_d, cpLength*sizeof(unsigned int));
cudaMalloc((void**)&rp_d, rpLength*sizeof(uint2));

cudaMemcpy(cp_d, cp, cpLength*sizeof(unsigned int), cudaMemcpyHostToDevice);
cudaMemcpy(rp_d, cpu_rp, rpLength*sizeof(uint2), cudaMemcpyHostToDevice);
cudaMemcpy(val_d, val, vLength*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(b_d, b, n*p*sizeof(float), cudaMemcpyHostToDevice);

// Calculate dimensions for GPU device.
dim3 grid; dim3 block;
grid.x = (unsigned int)(sqrt((float)n/2.0f)/BLOCK_SIZE + 1);
grid.y = (unsigned int)(sqrt((float)n/2.0f)/BLOCK_SIZE + 1);
block.x = BLOCK_SIZE; block.y = BLOCK_SIZE;

cudaEventRecord(start, 0);

```

```

// Kernel call.
mul_multipleblocks_BCRS<<<grid, block>>>(val_d, b_d, x_d, rp_d, cp_d, m, n);

// "Record" the stopping of this EVENT - i.e. return from kernel call.
cudaEventRecord(stop, 0);  cudaEventSynchronize(stop);

// Get the amount of time elapsed (in milliseconds) and DESTROY the CUDA
// timer objects.
cudaEventElapsedTime(&time, start, stop);  cudaEventDestroy(start); .
cudaEventDestroy(stop);

// Retrieve results pointed by 'x_d'
cudaMemcpy(x, x_d, m*p*sizeof(float), cudaMemcpyDeviceToHost);

// Free Memory - CPU.
delete [] cpu_rp;
// Free Memory - GPU.
cudaFree(val_d);
cudaFree(b_d);
cudaFree(x_d);
cudaFree(rp_d);
cudaFree(cp_d);
}

```

C/C++ Matrix Class

```

template <class T> class Matrix {
private:
    T *data_;

    // Number of Rows and Cols.
    unsigned int m_; unsigned int n_;

    // Matrix Compression Format.
    FORMAT_TYPE type_;

    // I-Index (e.g. ROW "pointer" in CRS).
    unsigned int *rowptr_;

```



```

// J-Index (e.g. COLUMN "pointer" in CRS).
unsigned int *colind_;

// First Index into Sub-Block for BCRS (i.e. 2x2 sub-blocks) Format.
unsigned int *nzptr_;

// Non-Zero(s) from ORIGINAL matrix for CRS (i.e. 1x1 Sub-Blocks).
T *val_;

// The LENGTH of 'val', 'colind_', 'rowptr_', and 'nzptr' Vectors respectively.
unsigned int vLength_; unsigned int cLength_; unsigned int rLength_;

// Compute the total number of Non-Zeros in this matrix.
unsigned int numNNZ();

// Compress current matrix to CRS Format.(i.e. 1x1 Block.)
void compressCRS();

// Compress current matrix to BCRS Format.(i.e. 2x2 Block.)
void compressBCRS();

// CPU-Based Matrix-Vector Multiplication(s) using CRS, BCRS (2x2), and
// NO Compression.
void matVecMultCRS(T *b, T *x, unsigned int n);
void matVecMultBCRS(T *b, T *x, unsigned int n);
void matVecMultNONE(T *b, T *x, unsigned int n);
void matMatMult_NONE(T *b, T *x);

// GPU-Based Matrix-Vector Multiplication(s) using CRS, BCRS (2x2), and
// NO Compression.
void matVecMultCRS_GPU(T *b, T *x, unsigned int p);
void matVecMultBCRS_GPU(T *b, T *x, unsigned int p);
void matVecMultNONE_GPU(T *b, T *x, unsigned int p);

public:
// Create Matrix object from argument data of m-by-n dimensions.
Matrix(T **data, unsigned int m, unsigned int n);

```

```

// Create Matrix object from argument data of m-by-m dimensions.
Matrix(T *data, unsigned int m);

// Compress current data element (i.e. matrix), REMOVING the ORIGINAL
// Matrix elements.
// PARAM: type The Matrix Compression used (e.g. CRS).
void compress(FORMAT_TYPE type = CRS);

// Computes Matrix-Vector Product as defined by the current matrix compression
// format (if any).
// PARAM: b Right-Hand Side
// PARAM: x Solution Vector (holds solution)
// PARAM: n Length of Right-Hand Side Vector and Solution Vector
void matrixVectorMult(T *b, T *x, unsigned int n);
void matrixMultCPU(T *b, T *x);

// Computes Matrix-Vector Product as defined by the current matrix compression
// format (if any) for the
// GPU Device.
// PARAM: b Right-Hand Side
// PARAM: x Solution Vector (holds solution)
void matrixVectorMultGPU(T *b, T *x, unsigned int p);
void matrixMultGPU(T *B, T *C);

T& operator()(unsigned int i, unsigned int j);
void printCompressType();

~Matrix();

};

```

C/C++ Class Template (calls CUDA file with Kernels)

```
template <class T>
void Matrix<T>::matVecMultCRS_GPU(T *b, T *x, unsigned int p){
    if(val_ == NULL || rowptr_ == NULL || colind_ == NULL)
        throw MatrixException("Exception with GPU call!\n");

    float gTime = 0.0f;

    // Call kernel via C-Code interface.
    mul_multBlocks_CRS(val_, vLength_, rowptr_, rLength_, colind_, cLength_, b, x, m_,
        n_, p, gTime);

    cout << "GPU Execution Time: " << gTime << " (milliseconds).\n";
}
```

Derivations of Presented Equations:

Equation (4.5) to calculate the average number of non-zeros per row when using the Compressed Sparse Row (CSR) data compression format is detailed below.

The data-type utilized in this work is the single-precision float each of which is defined by 4-bytes. The assumption is an initial square matrix of $M \times M$ dimension so the number of non-zero elements for each row M is generated as a ratio subtracted from the GPU device global memory G_{mem} . The numerator of the ratio is 4 times the number of rows M plus 1, to account for even numbers of elements as well as 4-byte floats. The denominator is the number of rows M times 8 which defines the square of a single float – generating a ratio that is less than 1 and an average of length of a single row in the original matrix. The maximum of 1 or the generated average number of non-zeros per row is chosen as the result R_{NZ} since the value should at least be a placeholder for any equation that employees this computed value.

$$R_{NZ} \cong MAX \left\{ 1, G_{mem} - \frac{4 \times (M + 1)}{(4 + 4) \times M} \right\} \quad (4.5)$$

Equation (4.6) to calculate the number of blocks when using the Compressed Sparse Row (CSR) data compression format is detailed below.

Each calculated block of data input to the GPU, N_B , is partitioned such that a single warp (32 threads) is given for each row M . The number of blocks is a ratio such that the numerator is

the product of the number of rows M and the average number of non-zero elements per row R_{NZ} and the denominator is the total number of warps and SMPs for the GPU device multiplied by the number of threads per warp, 32.

$$N_B \equiv \frac{M \times R_{NZ}}{N_t \times N_w \times N_{smp}} \quad (4.6)$$

Equation (5.3) to calculate the total solution time for multiple CPU/GPU computing systems when using Compressed Sparse Row (CSR) data compression format is detailed below.

The estimated solution time for the multiple CPU/GPU computing system is adapted from the estimated time for the *single* CPU/GPU computing system which is detailed in equation (4.9.3) of chapter 4 given as T_{gpu} , and the cost of local CPU-GPU host and intra-nodal MPI communication defined as C .

The assumption is made that the single CPU/GPU computing system solution time estimation T_{gpu} and the number of active thread blocks per SMP N_{ATB} are already known. The naïve approach of computing the ratio of the single CPU/GPU solution time by the number of partitions P_{sd} must be modified to account for overhead of communications defined as C . Given each partition will produce an individual cost C , C is divided by the number of partitions of the original global domain P_{sd} - this result is multiplied by the sum of the number of active thread blocks per SMP N_{ATB} and the square root of the number of partitions P_{sd} represented in equation (5.3) as $(N_{ATB} + \sqrt{P_{sd}})$. Multiplying $(N_{ATB} + \sqrt{P_{sd}})$ by the communication ratio $\frac{C}{P_{sd}}$ generates an average cost of communication assuming a square matrix, thus the $\sqrt{P_{sd}}$ variable.

The estimated solution time of *multiple* CPU/GPU computing system is not complete until the rate of growth/decay is calculated using an exponent of the ratio of the total number of partitions to the number of active thread blocks represented in equation (5.3) as $e^{\frac{P_{sd}}{N_{ATB}}}$. Using the generated rate of growth/decay $e^{\frac{P_{sd}}{N_{ATB}}}$, the cost of communication will increase as the number of partitions increase and conversely will decrease as the number of active thread blocks increase.

$$T_{mult_gpu} \equiv \frac{T_{gpu}}{P_{sd}} \left[\frac{C}{P_{sd}} (N_{ATB} + \sqrt{P_{sd}}) \right] e^{\frac{P_{sd}}{N_{ATB}}} \quad (5.3)$$

Appendix B

The CUDA Kernels and C/C++ code for the execution of full candidate application discussed in chapters 4 and 5 are presented below.

Main-point-of-entry (fertm2d.cpp):

```
// Name      : fertm2d.cpp
// Author    : Richard Haney
// Version   : 1.1a
// Description : Simulated Resin Transfer Molding (RTM) such that the global solution is solved
//              using Finite Element Method (FEM) and is based on original FORTRAN
//              COMPOSE2D code by Dr. Ram Mohan and Dale Shires.
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "Fertm.h"
using namespace std;

int main(int argc, char **argv) {
    Fertm fertm_(argc, argv);
    int nfill = 0; double sumf = 0.0;
    long long int flops = 0; double soltime = 0.0;

    // initialize system
    fertm_.initialize();

    // preprocess
    fertm_.preprocess(nfill);

    // process/solve system
    fertm_.process(nfill, sumf, flops, soltime);

    return EXIT_SUCCESS;
}
```

Interface and implementation of parent RTM object (IRTM.h):

```
#ifndef IRTM_H_
#define IRTM_H_

#include <string>
#include <iostream>
#include <stdlib.h>
using namespace std;

// Interface for Resin Transfer Molding (RTM) to be used to in the simulation program.
class IRtm {
public:
    IRtm();
    virtual ~IRtm();
    virtual string get_filename() = 0;

    /*
     * Function initializes all value(s) to prepare for the execution of the RTM program
     * *****
     *
     * PLEASE NOTE: Function must be called FIRST!
     * *****
     */
    virtual void initialize() = 0;

    /*
     * Function executes preprocessing operations for RTM program returning the
     * initial volume filled.
     * *****
     * PLEASE NOTE: Function must be called AFTER the initialize() and BEFORE the
     * process()
     * *****
    */
};
```



```

    * @param numfill is the current number of filled nodes - assumed zero at this point
    * @return total volume filled in model
    */
virtual void preprocess(int &numfill) = 0;

/*
 * Function executes the processing operations for RTM program - solving the system.
 * *****
 * PLEASE NOTE: Function must only be called AFTER calling the preprocess()
 * function.
 * *****
 * @param num is the number of filled nodes
 * @param sumf sum filled/volume
 * @param flops number of floating-point operations
 * @param solve_time total time for execution – in milliseconds
 * @param verbose if true outputs verbose info.
 */
virtual void process(int &num, double &sumf, long long int &flops, double &solve_time,
    bool verbose = false) = 0;
};
#endif /*IRTM_H_*/

```

Sub-class of RTM (Fertm.h):

```

#ifndef FERTM_H_
#define FERTM_H_

#include "IRtm.h"
#include "FertmModel.h"
#include "FertmParser.h"
#include "CStopWatch.h"

```

```

#include "CircValidate.h"
#include "Write.h"

class Fertm : public IRtm {
protected:
    FertmModel model_; FertmParser parse_;
public:
    Fertm(int argc, char **argv);
    virtual ~Fertm();

    // Function returns the current filename of the input file being "solved" by this class.
    string get_filename();

    // Function returns the current "partitioned" filename being used for MPI-based
    // parallelism, if any
    string get_pfilename();

    // Function performs initialization operations such that the FERTM 2D program can
    // execute properly.
    void initialize();

    /*
    * Function executes all preprocessing operations for the FERTM 2D program to execute
    * properly.
    * @param numfill current number of filled nodes - assumed zero at this point
    * @return total filled volume after preprocessing
    */
    void preprocess(int &numfill);

    /*
    * Function executes the processing operations for the FERTM 2D RTM - solving the
    * system.
    * *****
    * PLEASE NOTE: Function must only be called AFTER calling the preprocess()
    * function.
    * *****
    * @param num is the number of filled nodes
    * @param sumf sum filled/volume
    * @param flops number of floating-point operations

```

```
* @param solve_time total time for execution – in milliseconds
* @param verbose if true outputs verbose info.
*/
void process(int &num, double &sumf, long long int &flops, double &solve_time,
             bool verbose = false);

};

#endif /*FERTM_H_*/
```

Appendix C

The algorithms from chapter 2 of this dissertation defining the LCM solution strategy, sparse matrix-vector, and the preconditioned conjugate gradient iterative solver for sparse symmetric positive definitive matrices.

Algorithm 2.1: Implicit Pure FE methodology for LCM Computation

- (For time step $n + 1$ and iteration m)
1. **REPEAT**
 2. **SET** $\{\Psi_i\}_m^{n+1}$ to $\{\Psi_i\}^n$ (save previous fill factor values)
 3. **CALL** assembleC for C_i (assembleC forms lump mass matrix)
 4. **CALL** assembleK for K_{ij} (assembleK forms stiffness matrix K)
 5. **CALL** assembleLoad on q_i (assembleLoad forms load vector q)
 6. **REPEAT**
 7. **SET** boundary conditions on K_{ij}
 (Modified load vector g)
 8. **SET** $\{g_i\}_m$ to $C_{ii}\{\Psi_i\}^n - C_{ii}\{\Psi_i\}_m^{n+1} + \Delta t\{q_i\}_m$
 (Where \hat{K}_{ij} is K matrix with boundary conditions applied)
 9. **SOLVE** $[\hat{K}_{ij}]_m \{P_j\}_m = \{g_i\}_m$
 (Compute new nodal resin fraction field using equation (4))
 10. **SET** $C_{ii}\{\Psi_i\}_{m+1}^{n+1} = C_{ii}\{\Psi_i\}^n - \Delta t[K_{ij}]\{P_j\}_m + \Delta t\{q_i\}_m$
 11. **IF** $\|C_{ii}\{\Psi_i\}_{m+1}^{n+1} - C_{ii}\{\Psi_i\}_m^{n+1}\| \leq \xi$ **THEN**
 12. **BREAK**
 13. **ELSE**
 14. **SET** $\{\Psi_i\}_m^{n+1}$ to $\{\Psi_i\}_{m+1}^{n+1}$
 15. **ENDIF**
 16. **UNTIL** mass resin convergence
 17. **UNTIL** all nodes are filled
-
-

Algorithm 2.2: Preconditioned conjugate gradient (solves $Ax = b$)

Input: Matrix A and load/force vector b

Output: solution vector x

1. **Set** $r_0 \Leftarrow b - Ax_0$
 2. **Set** $z_0 \Leftarrow M^{-1}r_0$
 3. **Set** $p_0 \Leftarrow z_0$
 4. **Set** $k \Leftarrow 0$
 5. **DO UNTIL CONVERGENCE**
 6. $\alpha_k \Leftarrow \frac{r_k^T z_k}{p_k^T A p_k}$
 7. $x_{k+1} \Leftarrow x_k + \alpha_k p_k$
 8. $r_{k+1} \Leftarrow r_k - \alpha_k A p_k$
 9. **IF** $(\|r_k - r_{k+1}\| \leq \xi)$ **BREAK**
 10. $z_{k+1} \Leftarrow M^{-1}r_{k+1}$
 11. $\beta_k \Leftarrow \frac{z_{k+1}^T r_{k+1}}{z_k^T r_k}$
 12. $p_{k+1} \Leftarrow z_{k+1} + \beta_k p_k$
 13. $k \Leftarrow k + 1$
 14. **END DO**
-
-

Algorithm 2.3: Sparse Matrix-Vector Multiplication (CSR Compression)

Input: Non-zero vector dat , load/force vector b , row pointer $rptr$, column indices $cidx$, and row length M

Output: solution vector x

1. **Set** $i \leftarrow 0$
 2. **Set** $j \leftarrow 0$
 3. **Set** $k \leftarrow 0$
 4. **DO WHILE** ($i < M$)
 5. **Set** $j \leftarrow rptr[i]$
 6. **Set** $k \leftarrow rptr[i+1]$
 7. **DO WHILE** ($j < k$)
 9. **Set** $x[i] \leftarrow x[i] + (dat[j] \times b[cidx[j]])$
 10. **Set** $j \leftarrow j + 1$
 11. **END DO**
 12. **Set** $i \leftarrow i + 1$
 13. **END DO**
-
-

Appendix D

This appendix contains TECPLOT visualized results of resin flow progression contours of the input unstructured meshes.

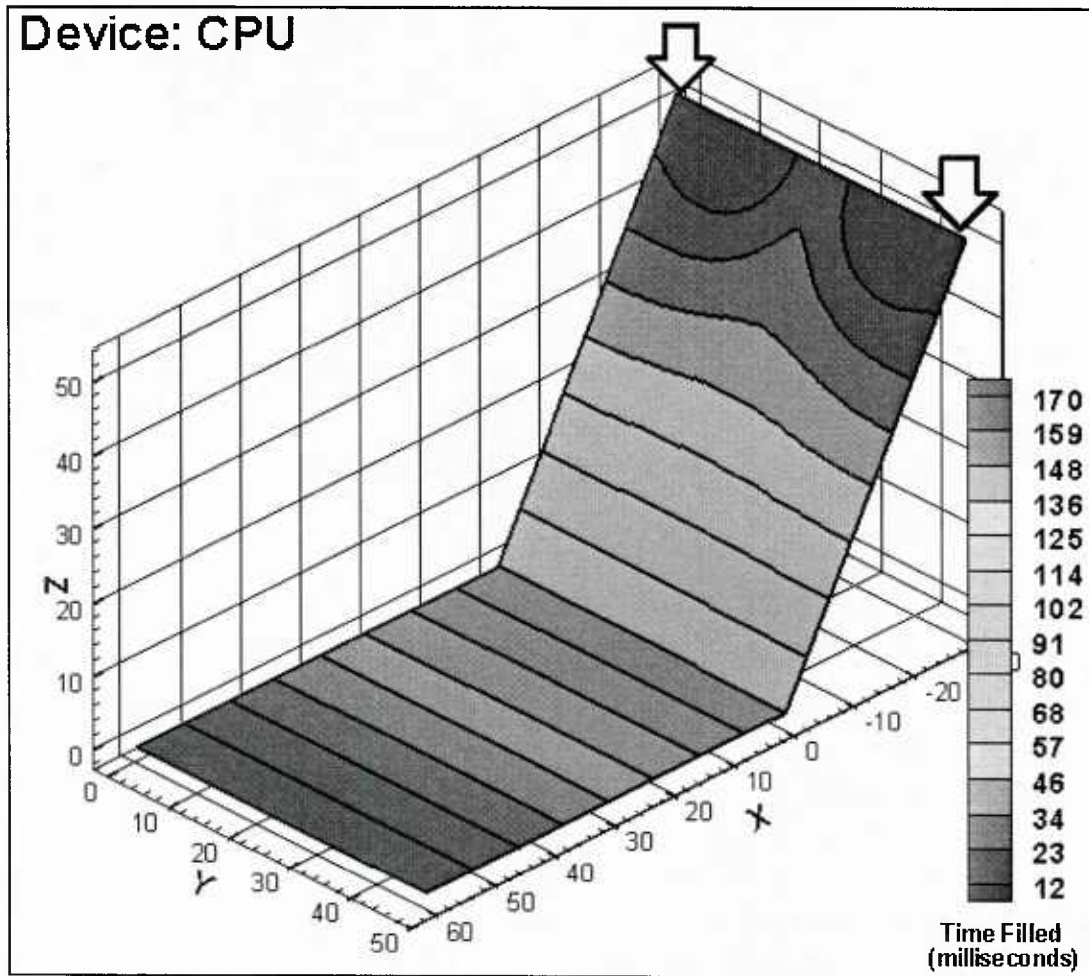


Figure 96. Time filled for unstructured mesh MA CPU-Only (System A)

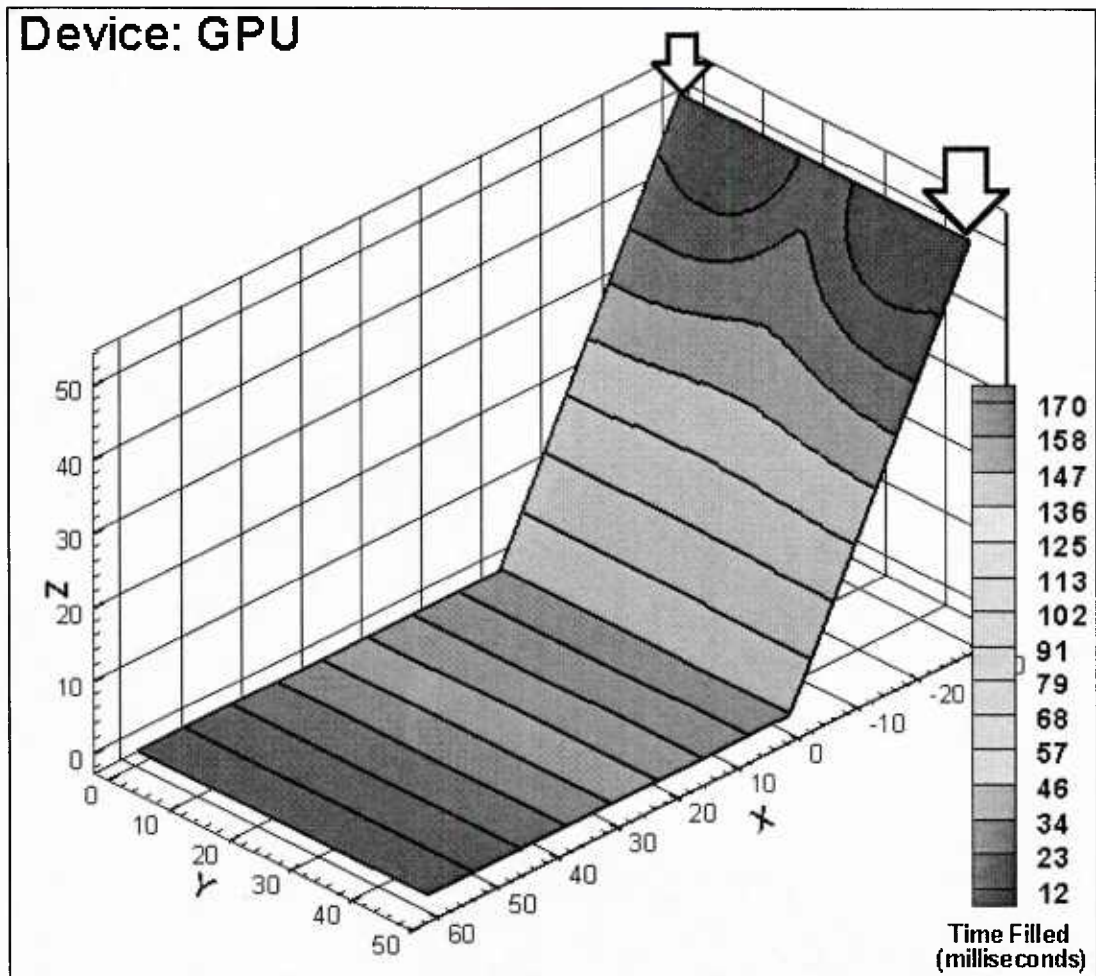


Figure 97. Time filled for unstructured mesh MA single CPU/GPU (System A)

The following are the time-filled TECPLOT images for validation using the 2D circular plate model that was compared to analytical solution.

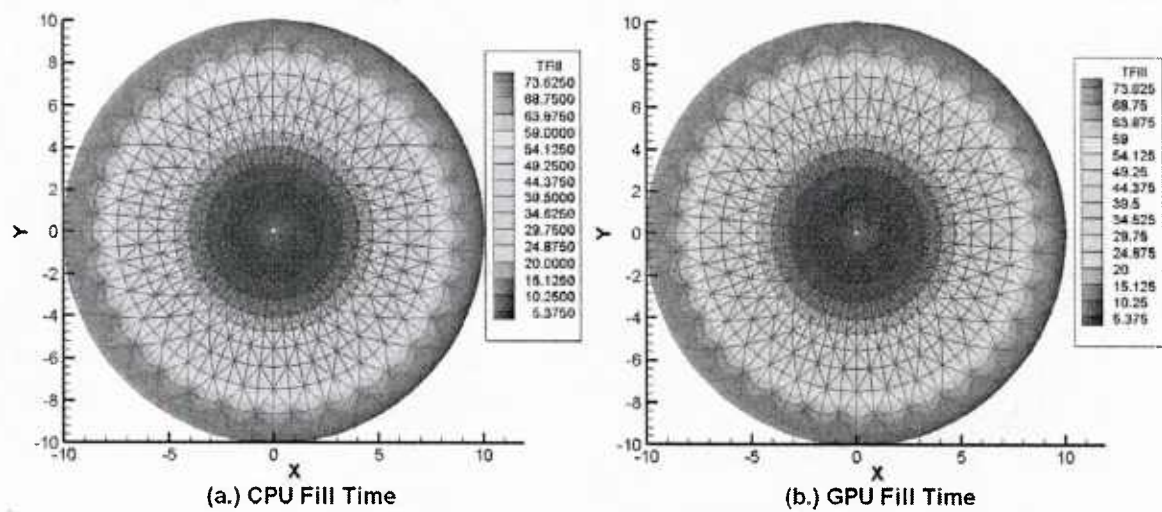


Figure 98. Time filled single CPU/GPU with circular plate (System A)

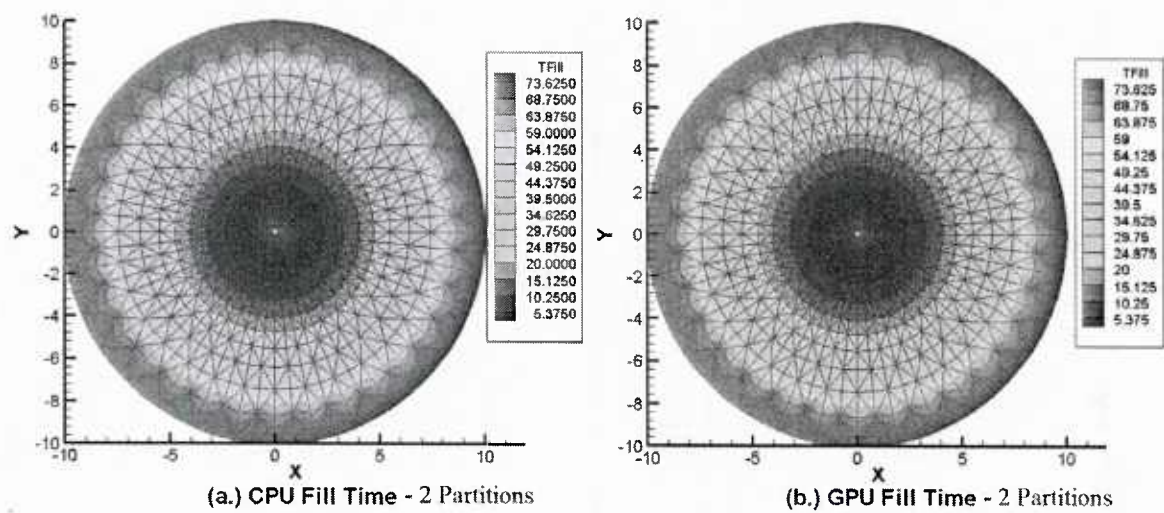
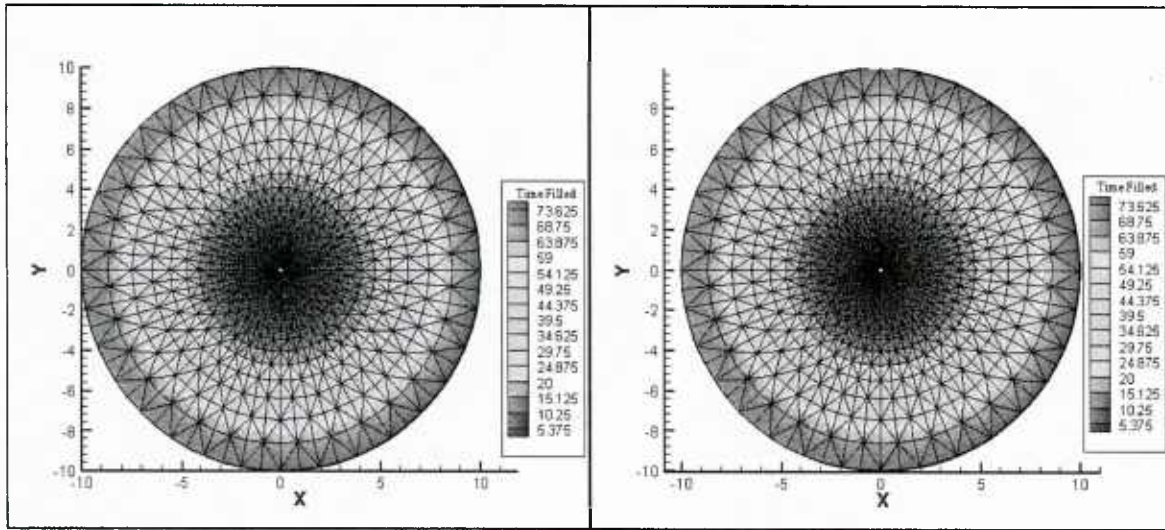


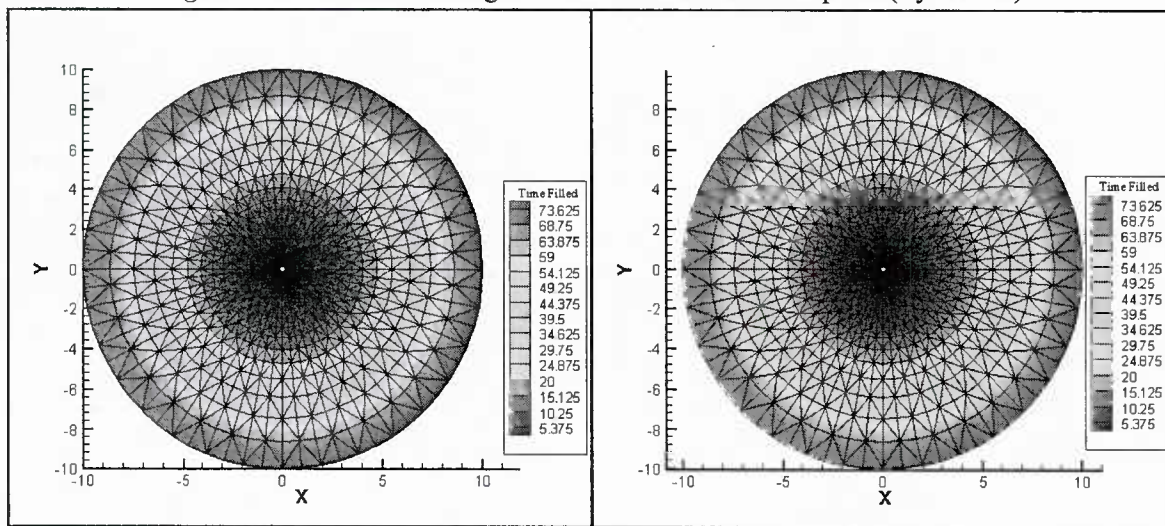
Figure 99. Time filled multiple CPU/GPU with circular plate (System A)



(a.) CPU Fill Time (ms.)

(b.) GPU Fill Time (ms.)

Figure 100. Time filled single CPU/GPU with circular plate (System B)



(a.) CPU Fill Time (ms.) - 2 Partitions

(b.) GPU Fill Time (ms.) - 2 Partitions

Figure 101. Time filled multiple CPU/GPU with circular plate (System B)

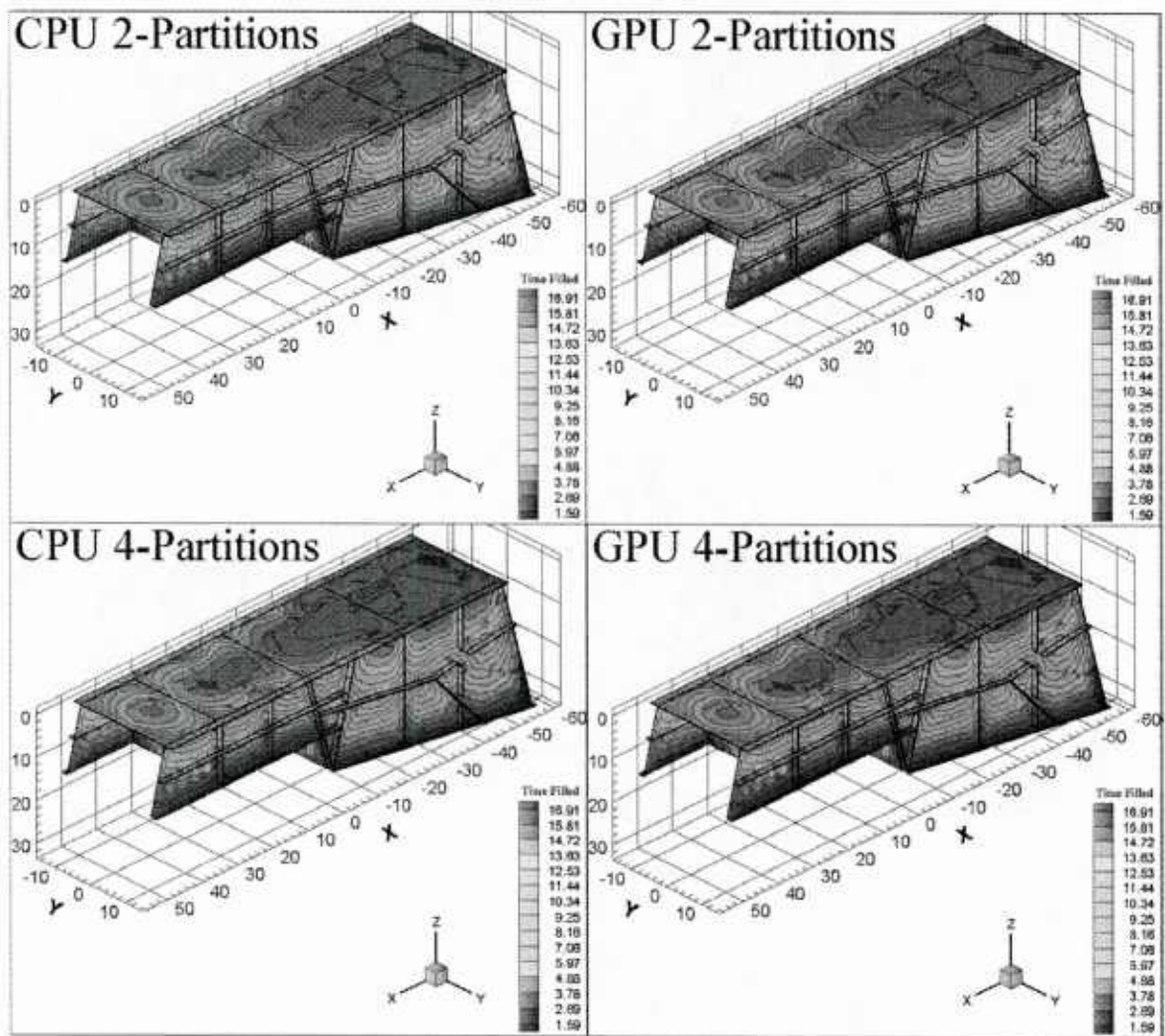


Figure 102. Input mesh model 10FT multiple partition time-filled comparison (System A)

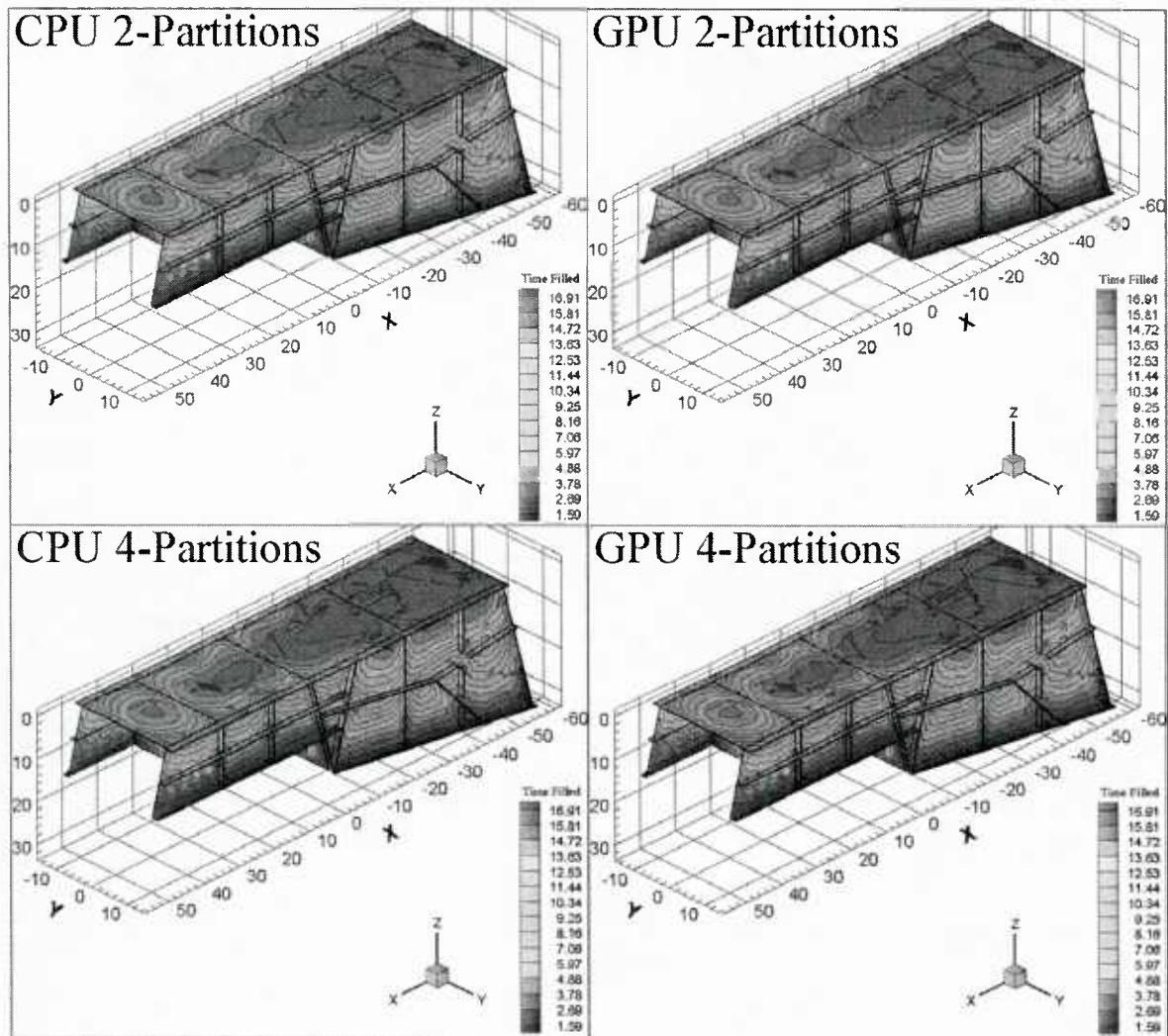


Figure 103. Input mesh model 10FT multiple partition time-filled comparison (System B)

B-3 Probabilistic Analysis of Property Uncertainties using Resin Infusion Flow Modeling and Simulations

Physics based flow modeling provides an effective way to simulate the resin infusion process in liquid composite molding processes for polymer composite structures. These are effective to provide optimal injection time and locations for given process parameters of resin viscosity and preform permeability prior to resin gelation. However, there could be significant variations in these two parameters during actual manufacturing due to differences in the resin batches, mixes, temperature, ambient conditions for viscosity; in the preform rolls, compaction, etc., for permeability. Research to understand the influence of uncertainties in these parameters on the resin infusion time was initiated via a probabilistic modeling methodology using resin flow modeling and statistical analysis. Project activities and discussions from this work are presented next.

Preform and Resin property uncertainties, role, and their effect in liquid composite process flow modeling

Authors: Ram V Mohan, Henok Shiferaw, Vinaya Kelkar, Ajit Kelkar

Published Paper: SAMPE 2012, Long Beach, CA, Paper 3230

ABSTRACT

Physics based flow modeling provides an effective way to simulate and understand the resin infusion process in liquid composite molding processes and its variants. These are effective to provide optimal injection time and locations for given process parameters of resin viscosity and preform permeability prior to resin gelation. However, there could be significant variations in these two parameters during actual manufacturing due to differences in the resin batches, mixes, temperature, ambient conditions for viscosity; in the preform rolls, compaction, etc., for permeability. The influence of uncertainties in these parameters on the resin infusion time is investigated using resin flow modeling and statistical analysis. Application of the process flow modeling and statistical analysis to understand the effect of preform and resin property uncertainties is demonstrated via a composite helicopter prototype part processed via vacuum assisted resin transfer molding. The probabilistic modeling methodology resulted in confidence envelopes to determine the probability for successful resin infusion prior to gelation, and estimate resin infusion time for any combination of viscosity and permeability. The effectiveness of these confidence envelopes to determine the probability for resin infusion success and

estimate the infusion time without a need for additional simulations and its usefulness for composite manufacturing engineers and technicians is presented.

INTRODUCTION

The manufacturing cycle time during resin infusion in liquid composite molding (LCM) of composite structural parts is influenced by various process and material conditions that include location of injection gates, fiber preform permeability, and resin viscosity. Successful infusion of dry fiber preform in LCM processes such as resin transfer molding (RTM) and its variants is one of the most complex and critical stages, and directly impacts the process performance and final quality of the part. Extensive effort has been conducted over the years on process flow modeling simulations for LCM processes and has been applied to several prototype developments [1,2,3,4,5]. Most of the process flow modeling approaches is based on deterministic models employing finite element based approaches for the space discretization. One such deterministic modeling methodology is an implicit transient approach based on transient mass conservation of the resin that has been validated and successfully demonstrated for very large scale simulations [5,6,7].

Deterministic physics based process flow modeling and simulations enable the down selection of optimal process parameters such as the injection gate locations. By analyzing various process scenarios through virtual simulations, effects of material and mold configurations can be thoroughly analyzed, even for a complex part. These deterministic process modeling simulations enable to study the effects of material, process and mold modifications, and obtain an optimal injection condition. This optimal condition can be subsequently employed during actual manufacturing process. Resin progression for the optimized injection condition and associated expected resin infusion time are based on specific values of key process parameter variables (fiber preform permeability and resin viscosity) that significantly influence the success of resin infusion. However, day to day, and batch to batch variations in the fiber preform rolls and preform layup differences can lead to variations in the fiber preform permeability. Similarly variations in the resin batches, ambient conditions, etc., can lead to differences in the resin viscosity. For a given composite part and mold configuration, injection gate conditions, any differences during actual manufacture of these two key process parameters can lead to significant variations from the modeling predictions. Deterministic modeling of all such variations of these two key process parameters would require significantly large number of flow modeling analysis. Variations in these two key parameters can be examined through statistical analysis, and provides an effective way to analyze their uncertainties, and develop confidence envelopes based on infusion time obtained from the simulations, and subsequently define the probability for successful infusion prior to resin gelation.

The present work employs a statistical analysis approach to analyze influence of uncertainties in these two process parameters of resin flow infusion. The statistical examination is built upon process flow modeling and simulations, and statistical analysis techniques. The influence of uncertainties in two key process parameters of resin viscosity and preform permeability on the resin infusion time is investigated. Resin infusion time output results for variations in the preform permeability and resin viscosity for given injection conditions are utilized to develop confidence envelopes using the calculated Cumulative Density Function (CDF). The obtained CDF is used to determine the probability for completion of resin infusion prior to physical resin gelation time.

The application of this methodology for simultaneous variations of these two key processing parameters in LCM processes is presented, and demonstrated for a composite helicopter prototype part. The probabilistic modeling methodology resulted in confidence envelopes to determine the probability for successful resin infusion prior to gelation, and estimate infusion time for any combination of viscosity and permeability for a composite part and associated injection conditions. The effectiveness of these confidence envelopes to determine the probability for resin infusion success and estimate infusion time without a need for additional simulations is demonstrated.

Present paper is organized as follows. A brief discussion of the process flow modeling methodology is described. This is followed by discussions on the application of the process modeling methodology and correlation with actual process observations for a composite helicopter prototype part processing. This composite part configuration also provides the demonstration application for the statistical analysis and development of the confidence envelopes. Details of the statistical analysis, development of confidence envelopes and results are discussed next. The probabilistic modeling methodology though illustrated with a demonstrative composite part configuration is applicable for other composite structures and provides an effective method to analyze and understand the effect of variations in resin viscosity and preform permeability. Furthermore, the developed confidence envelop provides manufacturing engineers and technicians a quick tool for evaluating the potential for successful resin infusion for any composite process run using the associated resin viscosities (for example, obtained through resin viscosity sampling) and preform permeabilities on any given day of manufacture. Examples of such scenarios are illustrated for this demonstrative composite prototype part configuration.

Resin Infusion flow modeling

Process Modeling Methodology

Resin mass conservation and infusion flow models in LCM processes address the macroscopic transient resin flow infiltration through a complex fiber preform. Resin infusion flow modeling

method used in the present work employs a transient resin mass conservation equation coupled with the Darcian flow behavior (momentum conservation) in conjunction with a pure finite element methodology that is used for tracking resin progression inside a complex mold cavity, representing the net-shape composite structural part. The presence of the resin in an Eulerian mold cavity is tracked using a state variable Ψ , defining the resin infused state of the region. The state variable Ψ varies between 0 and 1. The value of the state variable is 1 in the completely resin infused regions of mold cavity and 0 in the non-infused regions of mold cavity. The pressure gradient in the partially filled regions where the value of the state variable is between 0 and 1 is taken to be negligible. The integral form of the transient mass conservation equation is thus given by

$$\frac{d}{dt} \int_{\Omega} \Psi d\Omega = \int_{\Omega} \nabla \cdot \left(\frac{\bar{K}}{\mu} \nabla P \right) d\Omega \quad [1]$$

where, \bar{K} is the permeability tensor, μ is the resin viscosity, P is the pressure field, and Ψ is a state variable representing the infused state of the resin. Further details are available in references [5] and [6]. The permeability tensor is a second order tensor with four terms, with three of them unique, in most aerospace structures made of thin composite preform layers, where the flow velocity is primarily in the in-plane directions. Such thin composite structural configuration and analysis has been employed for the composite prototype part configuration.

Transient resin flow progression in the fiber preform geometry is analyzed through finite element geometry discretization via a pure finite element methodology [5] that is based on the above transient mass conservation equation. The state variable Ψ ($0 \leq \Psi \leq 1$) represents the infused state of a mold cavity region. $\Psi = 0$, represents the non-infused regions of dry fiber preform during transient flow. $\Psi = 1$, represents fully infused regions of dry fiber preform. Applying the Galerkin weighted residual formulation to the transient mass conservation equation, and introducing the finite element approximations for both state variable Ψ , and pressure field P , leads to a discretized system of equations given by

$$C\dot{\Psi} + KP = q \quad [2]$$

In equation 2, C is the mass matrix representing the pore volume, K is the stiffness matrix associated with the pressure field. The time derivative term is discretized using equation 3.

$$\dot{\Psi} = \frac{\Psi_{n+1} - \Psi_n}{\Delta t} \quad [3]$$

In the above equation, Δt is the time step size for the transient problem, and q is the force vector representing the injection conditions. The boundary conditions are given by

$$\frac{\partial P}{\partial n} = 0 \text{ at mold walls,}$$

$$P = 0 \text{ at flow front, and}$$

$$P = P_0 \text{ prescribed pressure at inlet} \quad [4]$$

or

$$q = q_0 \text{ prescribed flow rate at inlet,}$$

where P_0 and q_0 represent prescribed pressure and flow rate at the inlet(s), respectively. Initially, at time $t=0$,

$$\Psi = 1 \text{ at the inlet and}$$

$$\Psi = 0 \text{ elsewhere.} \quad [5]$$

The pure finite element methodology iteratively solves for the state variable, Ψ , that defines the infusion state and the associated pressure until complete mass conservation is achieved at each time step. A resin infusion flow modeling code based on the above methodology for a thin shell 2.5D composite flow configuration is employed for the flow modeling simulations presented in the present work.

Process Flow Modeling Application to a Composite Helicopter Prototype Part

Process flow modeling analysis was employed to obtain optimal line based infusion configuration for a complex, composite helicopter prototype part consisting of bi-directional carbon preform and epoxy resin. Figure 1 presents the geometry and computational finite element mesh for a 2.5D thin shell resin infusion analysis. The complex part is approximately $1.016 \text{ m} \times 0.635 \text{ m}$, with a compacted preform thickness of 0.18 cm . The permeability of the plain weave, bi-directional preform employed is $22.58\text{E-}10 \text{ m}^2$. The viscosity for the epoxy resin used in the prototype part processing is 0.35 PaS . Resin infusion is driven by an atmospheric vacuum pressure differential of 98.2 KPa . Six different line based resin injection schemes were analyzed for the resin flow progression and total infusion time. Figure 2 presents the resin infusion progression from these process flow modeling simulations. The temporal resin progression contour is coded from blue to red showing the time progression of resin infusion. Table 1 presents the predicted resin infusion time obtained from flow modeling in each line based injection scheme. While the line based injection scheme B and C predicted higher resin infusion times, injection configurations E and F predicted lower resin infusion times.

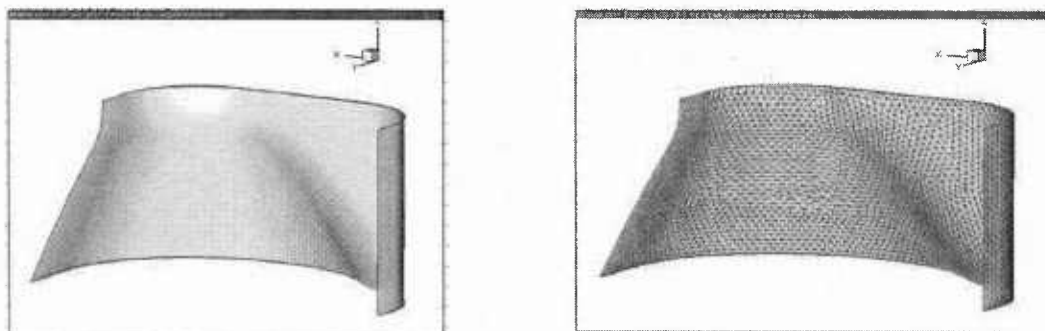


Figure 1. Composite prototype part and finite element mesh.

For the prototype part infusion, the placement of resin feed line along the curved part edge presents practical difficulties. Injection scheme F required two resin feed lines. Based on these

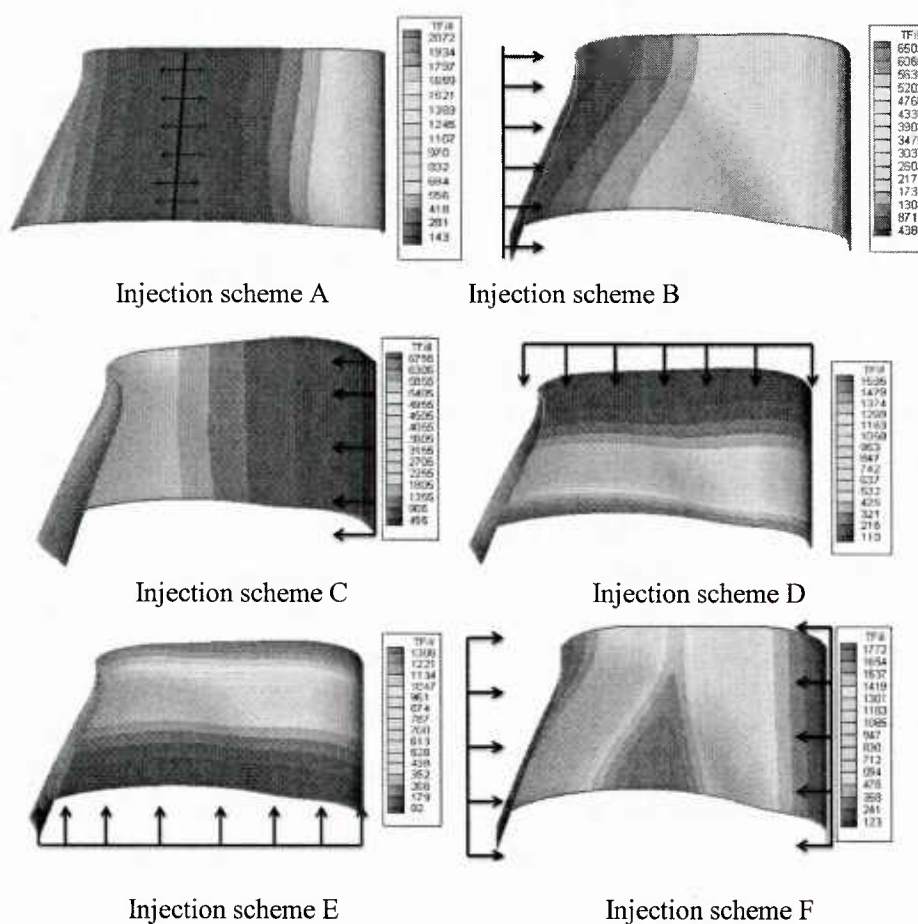


Figure 2. Transient resin flow progression with different line injection schemes.

practical considerations and predicted resin infusion times obtained, resin injection scheme - A

was selected for the prototype fabrication of this helicopter composite panel and employed in the actual prototype processing process. Flow progression observations from the prototype processing of this composite part are presented and discussed next.

Table 1. Predicted resin infusion time.

Model	Total Fill Time (minutes)
Injection scheme A	37
Injection scheme B	120
Injection scheme C	115
Injection scheme D	28
Injection scheme E	23
Injection scheme F	31

Prototype Part infusion and simulation comparison

Vacuum based resin infusion for this composite prototype part was setup in our processing laboratory employing line injection scheme - A via the H-VARTM process [8]. Resin infusion was performed with an aerospace grade epoxy resin with a flow viscosity of 0.35 PaS . Resin progression during this prototype part processing was observed and the resin infusion time was recorded. Figure 3 presents a snapshot of the resin progression during the infusion. The dark lines marked show the fully saturated resin front location at an instant of time that is used for simulation comparisons.

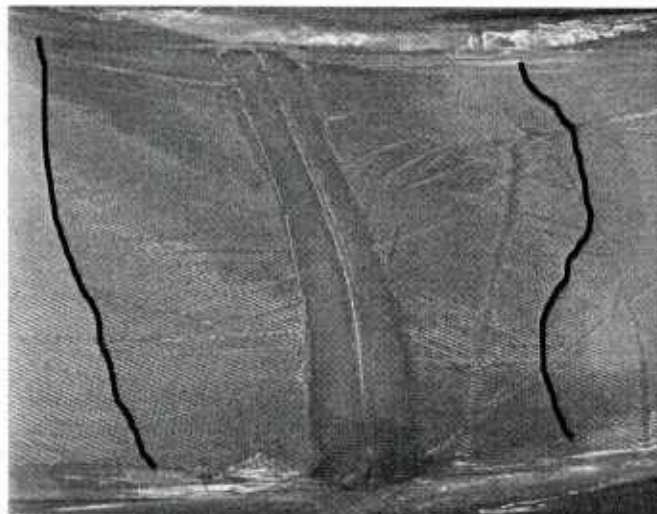


Figure 3. Snapshot of resin progression during prototype part infusion.

The infusion time for this prototype processing was about 45 minutes and is in good comparison with the predicted infusion time from prior process flow modeling simulations. The following were however observed during this prototype part resin infusion process. During the resin infusion, air bubbles were observed in the infused region and resin feed line. This required the resin feed line to be closed, check the sealing to ensure that there are no vacuum leaks; block the resin feed line on the preform midway, and restart the infusion process. Air bubbles, resin feed line modification, and vacuum leaks significantly influence resin progression and increase the infusion time. This corrective action resulted in an altered and new filling pattern that deviated from the resin progression from simulations shown for the injection scheme - A in figure 2.

To understand the effect of this change in resin infusion and to check the capability flow modeling simulations to capture and emulate these variations, the injection boundary condition in the selected injection configuration - A was modified. The injection pressure differential in the flow modeling simulations was modified to match the conditions during actual prototype processing. This was emulated in the process flow modeling simulations through a pressure drop that varied only half way through the feed line from the injection end. This modified injection condition employed in the process flow modeling resulted in a new predicted infusion time of 47 minutes. This concurs well with the infusion time obtained for the actual composite prototype part processing. Furthermore, the simulated resin front progression based on the above modified injection conditions showed reasonably excellent agreement to that obtained from the actual composite prototype part processing. The marked flow front line in black in figure 4 represents the resin saturated region from the simulations at the same instant of time as shown in the prototype part processing in figure 3. This flow front snapshot and resin progression patterns from the simulations demonstrated good agreement with that of the actual progression for this composite prototype part. Clearly, process flow modeling simulations emulate and capture the process variations during resin infusion if accounted correctly.

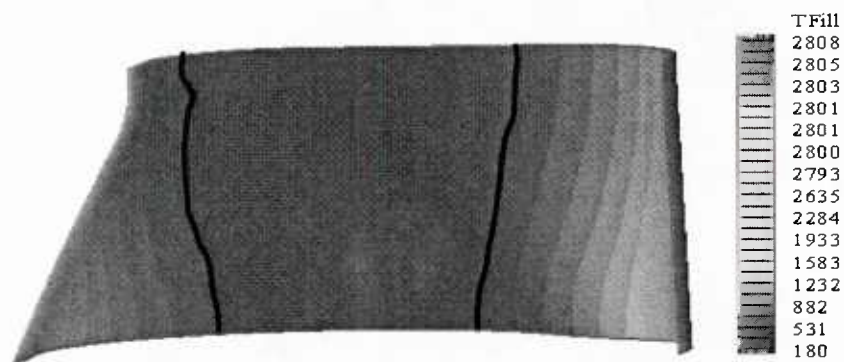


Figure 4. Transient resin front progression with modified injection condition.

The success of infusion on any given day during the production process however would depend on the completion of resin infusion prior to gelation. The infusion time for any given injection conditions are influenced by the variations in the preform permeability and resin viscosity. The preform permeability can show uncertainties and variations from batch to batch; variations during layup, vacuum pressure differential, etc. Resin viscosity can show variations and uncertainties from one manufacturing process run to the next. The understanding of these variations and obtaining predicted infusion time for the conditions on any particular day will require another process flow modeling simulation, preferably in real time that could become realistically impractical. A probabilistic analysis of these preform and resin preform uncertainties, their role, effect, and potential utilization of such analysis during actual day to day manufacturing are presented next.

Probabilistic analysis of property uncertainties

Composite process flow modeling simulations enable analysis of various process injection scenarios for optimal injection gate locations; understand subsequent effects of material and mold variations to ensure complete infusion prior to gelation. For a given mold and injection scheme configuration, resin infusion time are dependent upon the specific values of key process variables, preform permeability and resin viscosity and impact successful resin infusion. During repeated manufacturing, day to day, and batch to batch variations in fiber preform rolls and preform layup differences lead to variations and uncertainties in preform permeability. Similarly, resin batch variations and ambient conditions could cause variations and uncertainties in resin viscosity. Both critically influence the resin infusion time and success of resin infusion. Statistical variations and distributions of these two key process parameters were studied. A number of process flow modeling analysis for the composite helicopter prototype part, one for each of the statistically distributed permeability and resin viscosity parameter values were performed to obtain the corresponding resin infusion time. This provided a distribution of expected resin infusion time for the associated distributions of variations and uncertainties in the preform permeability and resin viscosity. Cumulative Density Function (CDF) of the obtained resin infusion time distribution was employed to obtain confidence envelopes. All statistical data analysis was performed using SPSS [9]. CDF and confidence envelopes determine the probability for completion of infusion before a specified resin gelation time for any combination of resin viscosity and preform permeability. Details, results and discussions from the probabilistic analysis for the prototype composite part are presented next.

Composite Helicopter Part – Probabilistic Analysis of Permeability and Viscosity Uncertainties

Statistical distribution of permeability and resin viscosity to understand the uncertainties were obtained employing statistical analysis package SPSS with a viscosity range of $\pm 20\%$ of a mean resin viscosity value of 0.35 PaS , and a $\pm 50\%$ variation from the mean permeability of $22.58 \text{E-}10 \text{ m}^2$. Statistical analysis of the resin viscosity variation individually (keeping permeability constant) showed a linear variation in the computed resin infusion time. As expected, the resin infusion time increased with increase in viscosity.

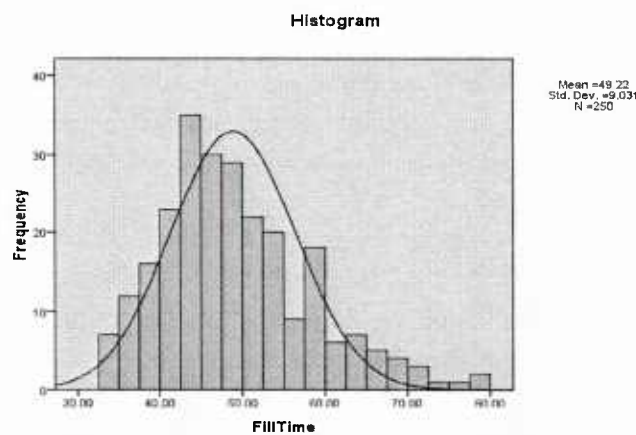


Figure 5. Resin infusion fill time distribution due to uncertainties in permeability and viscosity.

Analysis of permeability variation individually showed a non-linear decrease in resin infusion time with increasing permeability. Based on this, five different viscosity values within the resin distribution and fifty different permeability values within the permeability distribution were selected for the statistical analysis of the coupled permeability and viscosity variations and uncertainties. This permitted a good statistical distribution of resin infusion time to be obtained to understand the coupled uncertainties in resin viscosity and preform permeability.

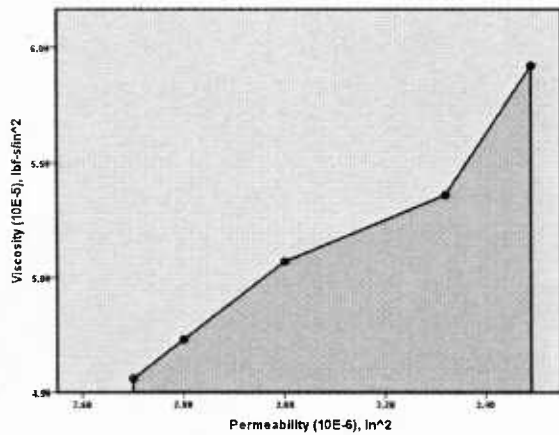
Figure 5 presents the statistical distribution of the resin infusion time obtained from the process flow modeling simulations. The coupled permeability and resin viscosity uncertainty shows a skewed distribution though viscosity and permeability had a normal distribution. This confirms the non-linear effect from coupled viscosity and permeability uncertainties. Statistical analysis of the resin infusion data for this composite helicopter part and "line injection scheme - A" showed that viscosity contributed to 25% of the variation in the resin infusion time, while permeability contributed to 68% of the variations in the resin infusion time. Cumulative Distribution Function (CDF) and probability for the resin infusion fill time (FT) to be less than a specified resin

gelation time (GT) were obtained from the normalized distribution of the resin infusion fill time. Viscosity and permeability ranges for 80% and 95% probability/confidence level for the resin infusion fill time (FT) to be less than the resin gel time (GT) is presented in Table 2. A resin gel time (GT) of 55 minutes was used.

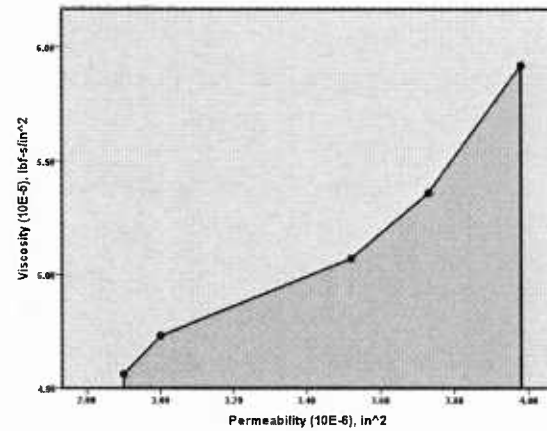
Table 2. Viscosity – Permeability range for successful infusion.

Viscosity ($lbf \cdot s/in^2 / PaS$)	Permeability (in^2 / m^2)
80% Confidence Level	
$4.56 \times 10^{-5} / 0.315$	$K \geq 2.70 \times 10^{-6} / 17.80E-10$
$4.73 \times 10^{-5} / 0.327$	$K \geq 2.80 \times 10^{-6} / 18.06E-10$
$5.07 \times 10^{-5} / 0.350$	$K \geq 3.00 \times 10^{-6} / 19.35E-10$
$5.36 \times 10^{-5} / 0.370$	$K \geq 3.32 \times 10^{-6} / 21.42E-10$
$5.92 \times 10^{-5} / 0.409$	$K \geq 3.49 \times 10^{-6} / 22.52E-10$
95% Confidence Level	
$4.56 \times 10^{-5} / 0.315$	$K \geq 2.90 \times 10^{-6} / 18.71E-10$
$4.73 \times 10^{-5} / 0.327$	$K \geq 3.00 \times 10^{-6} / 19.35E-10$
$5.07 \times 10^{-5} / 0.350$	$K \geq 3.52 \times 10^{-6} / 22.71E-10$
$5.36 \times 10^{-5} / 0.370$	$K \geq 3.73 \times 10^{-6} / 24.06E-10$
$5.92 \times 10^{-5} / 0.409$	$K \geq 3.98 \times 10^{-6} / 25.68E-10$

Figure 6 presents the viscosity and permeability ranges that would ensure 80% and 95% probability and confidence of resin infusion fill time (FT) to be less than the gelation time (GT) for successful infusion. Any permeability and viscosity combination that is within the shaded region of figure 6 would indicate the associated probability and confidence level of successful infusion. For example, permeability – viscosity combination within the 95% confidence interval shaded region in figure 6(a) would indicate a 95% probability of successful infusion prior to gelation. This confidence for successful infusion can be obtained without a need for additional real time simulations with the associated viscosity and permeability on the actual day of manufacture.



(a) 80% confidence level



(b) 95% confidence level

Figure 6. 80% and 95% confidence envelope for permeability and viscosity uncertainties.

The expected infusion time can also be estimated from the probability and the associated resin infusion fill time data obtained from the statistical analysis [10]. For large complex parts, the computing time for the flow analysis can be significant and preclude any real time simulations. Probabilistic analysis and confidence envelopes as discussed in this paper can be developed for any composite part and processing configuration prior to actual full scale manufacturing. This not only enables an understanding of the effect of permeability and viscosity uncertainties prior to full scale manufacture. It also provides an effective means to determine the probability of resin infusion success based on the conditions and variables during actual manufacture to estimate the expected resin infusion time without a need for additional full scale process flow modeling simulations. The potential application of the developed confidence envelope is demonstrated next.

Application of Confidence envelope

The probability analysis methodology for understanding the uncertainties discussed in this paper can be applied for any composite part processing application. The technique can be expanded and is applicable even if there are varying regions of permeability within a complex part to further consider their associated uncertainties. The developed confidence envelopes provide composite manufacturing engineers and technicians an easy to use analytical capability to determine the probability and confidence of successful infusion prior to resin gelation and estimate infusion time for a combination of preform permeability and resin viscosity on any given day of manufacture. This can be obtained without a need for additional flow modeling simulations, and are effective to not only understand the effect of these parameter variations, but also estimate the process success. This is more desirable and efficient in large complex

composite parts where the process flow simulations can take significant computational time. Two demonstration scenarios of the application of the confidence envelopes developed for the prototype composite helicopter part are illustrated next.

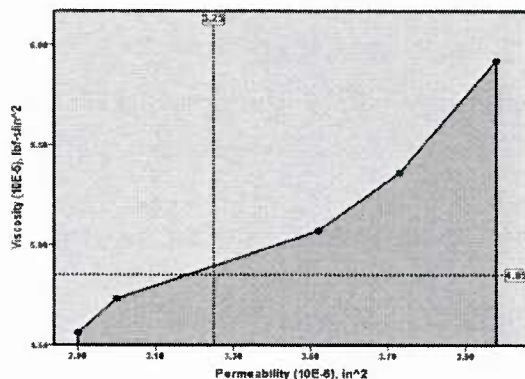
Processing Scenario 1

A composite manufacturing engineer/technician collects resin viscosity data and obtains the preform permeability of the batch of preform used. On a given day, these values of resin viscosity and permeability are 0.366 PaS and $20.97\text{E-}10 \text{ m}^2$ respectively. This resin viscosity, permeability data point is identified to be within the 95% confidence region for the prototype composite helicopter part as shown in figure 7-a. In addition, an estimation of the expected resin infusion time can also be obtained from a plot of the resin infusion time and associated probability.

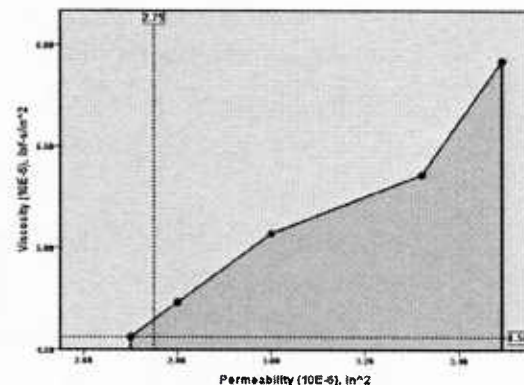
Processing Scenario 2

On a different day of manufacture, resin viscosity of the resin batch used is 0.389 PaS and the permeability of the preform batch is $17.74\text{E-}10 \text{ m}^2$. This viscosity, permeability combination is within the 80% confidence region for the prototype composite helicopter part, but well outside the 95% confidence region.

Thus by using the pre-developed confidence envelopes associated with any composite part, a composite manufacturing engineer and technician can estimate the confidence of successful resin infusion prior to gelation on any given day of manufacture using the associated preform permeability and resin viscosity values. Further, using their experience and knowledge over several days of production, composite manufacturing engineers and technicians can decide if corrective actions are needed prior to and during actual manufacture to ensure the success of manufacture on any given day.



a. Scenario 1 (95% confidence)



b. Scenario 2 (80% confidence)

Figure 7. Application confidence envelope during actual manufacture.

Concluding remarks

Process flow modeling simulations and analysis of resin infusion in LCM processes are effective to obtain resin infusion time and understand transient progression of resin enabling the determination of optimal injection conditions. However these simulated behaviors are based on the key process parameters of preform permeability and resin viscosity employed in the simulations. Significant variations in these two key process parameters can occur during the actual manufacture and during production cycle for a composite part. A probabilistic analysis approach for coupled effect of uncertainties of these two key process parameters using resin infusion process flow modeling simulations was presented. The applicability of the methodology was demonstrated for a composite prototype part processing employing vacuum based resin infusion. The computational modeling framework and probabilistic methodology that resulted in the development of confidence envelopes provides composite manufacturing engineers and technicians an estimate of the confidence for successful resin infusion prior to resin gelation, and expected resin infusion time on any given day of manufacture using the associated preform permeability and resin viscosity values. Furthermore, this can be obtained without a need for additional flow modeling simulation analysis using associated parameters on the day of manufacture. This provides an enabling and effective simulation based analytical capability for composite manufacturing engineers and technicians coupled with their experience over several infusion and processing runs to decide on corrective actions prior to, and during actual manufacture to ensure success of manufacture on any given day and part production run. The applicability of the confidence envelopes was demonstrated with illustrative scenarios based on a composite helicopter prototype part. The techniques and methods discussed can be extended for any composite structure and its processing with liquid resin infusion processes.

References

1. Bruschke, M. & Advani, S., "A Finite Element Control Volume Approach to Mold Filling in Anisotropic Porous Media." *Polymer Composites* 11(1990):390-405.
2. Troughu, F., Gauvin, R. & Gao, D., "Numerical Analysis of the Resin Transfer Molding Process by the Finite Element Method." *Advances in Polymer Technology* 12(1993):329-342.
3. Cai, Z., "Simplified Mold Filling Simulation in Resin Transfer Molding." *Journal of Composite Materials* 26(1992):2606-2630.
4. Lin, R., Young, W. & Lee, J., "Mold Filling and Cure Modeling of RTM and SRIM Processes." *Composite Structures* 27(1994):109-120.

5. Mohan, R., Ngo, N. & Tamma, K., "On a Pure Finite Element Methodology for Resin Transfer Mold Filling Simulations." *Polymer Engineering and Science* 39(1999):26-43.
6. Mohan, R., *et. al.*, "Advance Manufacturing of Large Scale Composite Structures: Process Modeling, Manufacturing Simulations and Massively Parallel Computing Platforms." *Advances in Engineering Software* 29(1998):249-264.
7. Mohan, R., *et. al.*, "Three Dimensional Resin Transfer Molding: Isothermal Process Modeling and Implicit Tracking of Moving Fronts for Thick Geometrically Complex Composites Manufacturing Applications." *Numerical Heat Transfer – Part A, Applications*. 35(1999):839-858.
8. Bolick, R., & Kelkar, A. *H-VARTM Processing*, US Patent Application Number: 12/361224, Publication Number; US 2009/0189320A1
9. SPSS Version 20, 2011, SPSS Inc., Chicago, IL.
10. Shiferaw, Henok. *Probabilistic Analysis of Property Uncertainties Using Resin Infusion Flow Modeling and Simulations – Resin Viscosity and Preform Permeability*. MS Thesis, North Carolina A&T State University, Greensboro, NC, 2011.

Statistical Analysis of Uncertainties in Deterministic Computational Modeling – Application to Composite Process Resin Infusion Flow Modeling

Authors: V. Kelkar, R. Mohan, H. Shiferaw, and A. Kelkar

North Carolina A&T State University

Greensboro, NC, USA

Journal Article: Communications in Statistics – Simulation and Computation

DOI: 10.1080/03610918.2013.815775

Abstract

Deterministic physics based flow modeling provides an effective way to simulate and understand the resin flow infusion process in liquid composite molding processes and its variants. These are effective to provide optimal injection time and locations prior to gelation for given process parameters of resin viscosity and preform permeability. However, there could be significant variations in these two parameters during actual manufacturing. This paper presents simulation based statistical analysis of uncertainties of these process parameters involved in the resin flow infusion. Two key process parameters, viscosity and permeability and their statistical variations are examined individually and subsequently in combination for their impact on the associated injection time. Values from statistical probability distribution of the process parameters were employed to find the solution space for this engineering application through deterministic physics based process flow modeling simulations. A bivariate confidence envelope was developed using the appropriate Cumulative Density Function (CDF) for a 95% probability of successfully completing resin infusion prior to physical resin gelation time. A logistic regression model for the influence of resin viscosity and permeability on the binary response of successful resin infusion is presented and conforms well to the sensitivity analysis inferences.

Keywords: Uncertainty quantification; Composite process flow modeling; statistical analysis; Deterministic computational modeling and parameter variations; logistic regression analysis;

Introduction

Liquid composite molding (LCM) processes such as resin transfer molding (RTM), vacuum assisted resin transfer molding (VARTM), and its variants are increasingly used as manufacturing processes for composite structures [1,2,3,4]. A schematic of the liquid composite molding process for composite structures is shown in figure 1. These processes are based on the impregnation of a net-shape dry fiber preform with a low viscosity polymeric resin. The polymeric composite part is then formed after the complete infiltration of the dry fiber preform prior to resin gelation and subsequent curing reaction. The manufacturing cycle time during resin infusion is influenced by various process and material conditions including the location of the injection gates, fiber preform permeability, and resin viscosity. Successful infusion of the dry fiber preform in liquid composite molding processes is one of the most complex and critical stages in the process and directly impacts the process performance and final quality of the part. Resin infusion has to be completed before physical resin gelation while the resin can still flow. This limiting time for a resin is characterized by its resin gelation time.

Deterministic physics based process modeling and simulations enable the down selection of optimum process parameters such as the injection gate locations. Extensive effort has been done over the years on process flow modeling simulations for resin flow infusion and has been applied to several prototype developments [5,6,7,8,9,10,11]. These deterministic simulation based optimal conditions can be subsequently employed during the actual manufacturing process. These optimal process conditions are based on specific values of key process parameters of fiber preform permeability and resin viscosity that significantly influence the success of resin infusion. However, day to day, and batch to batch variations in the fiber preform rolls and preform layup differences during manufacturing can lead to variations in the fiber preform permeability. Similarly variations in the resin batches and ambient conditions can lead to differences in the viscosity. Such variations in the key parameters can be examined through statistical analysis, and provides an effective way to analyze these key process parameter uncertainties, and develop associated confidence envelopes.

In the present paper, we describe a non-deterministic statistical simulation analysis approach to analyze the influence of uncertainties in the process parameters for resin flow infusion. The statistical examination is built upon deterministic computational physics based process flow modeling simulations, and statistical analysis techniques. Two key process parameters and their statistical variations are examined, initially based on individual parameters and subsequently as a combination of these two parameters. The data values from probability distribution of the key process parameters determine the modeling analysis solution space. A confidence envelope is subsequently developed using the calculated Cumulative Density Function (CDF). Statistical

CDF determines the probability for completion of infusion prior to a physical resin gelation time for variations in a real-valued random variable (for example, a key process parameter) with a given probability distribution. Such confidence envelopes are obtained for variations in a single parameter and coupled two parameters. Our approach thus employs non-deterministic statistical analysis in conjunction with deterministic physics based computational modeling and simulations. This provides an effective way to analyze and understand the effect of key parameter variations on the resin infusion success. The confidence envelopes obtained through statistical analysis provides an effective means to determine the probability for successful resin infusion prior to resin gelation time for the actual process parameters that can change. Further, the expected resin infusion time can be obtained without a need for additional full scale simulations. In engineering applications, due to inherent uncertainty and variation in materials and processes, model and simulation based uncertainty quantification within an appropriate statistical probabilistic framework is required. Present paper provides and demonstrates the application of such a statistical simulation analysis framework within the context of resin infusion flow modeling for liquid composites processing.

Deterministic Resin Infusion Flow Modeling

Resin mass conservation and process flow models address the macroscopic transient resin flow infiltration through a complex fiber preform in LCM processes. The deterministic resin infusion flow modeling method in the present work employs a transient resin mass conservation equation coupled with the Darcian flow behavior (momentum conservation) in conjunction with a pure finite element methodology that is used for tracking the resin progression inside a complex mold cavity representing the net-shape composite structural part. The integral form of the transient mass conservation equation solved during process flow infusion modeling is given by

$$\frac{d}{dt} \int_{\Omega} \Psi d\Omega = \int_{\Omega} \nabla \cdot \left(\frac{K}{\eta} \nabla P \right) d\Omega \quad (1)$$

where, K is the permeability tensor, η is the resin viscosity, P is the pressure field, and Ψ is a state variable representing the infused state of the resin inside the mold cavity. The permeability tensor is a second order tensor with four terms, three of them unique, in most aerospace structures made of thin composite preform layers. Transient progression of the resin through the fiber preform geometry is analyzed with a pure finite element methodology [9] that is based on the above transient mass conservation equation. Further details are available in references [9] and [10].

Optimization of injection locations based on these deterministic physics based modeling have employed genetic algorithms (GA), continuous sensitivity equation (CSE) analysis, and hybrid approaches (coupled GA and CSE) to determine optimal injection gate location in LCM

[12,13,14,15,16,17,18]. These methods obtain optimal injection gate locations based on minimizing the resin infusion time. Though these techniques allow the determination of optimal injection gate locations employing deterministic modeling and simulation of resin flow infusion, they do not address the stochastic, non-deterministic, statistical variations due to material, process setup; their influence on the resin infusion time and flow progression once the optimal injection gate location is obtained.

Non-Deterministic Statistical Analysis

In this work, effect of the uncertainties and variations in two key parameters (viscosity and permeability) on the infusion time and flow front progression for a vacuum based resin infusion process was studied.

Resin flow rate is governed by many process parameters, such as: a) Injection pressure, b) Mold temperature, complexity, and fiber reinforcement; c) Resin chemistry and rheology (resin viscosity), d) Permeability of the preform. For a given mold configuration (mold complexity) and injection conditions (injection pressure), resin flow behavior is heavily influenced by preform permeability and resin viscosity. The resin flow velocity through fiber preform given by Darcy's law (equation 2), defines the flow behavior in process flow models and governs the resin infusion or filling time (FT).

$$\vec{v} = -\frac{\overline{K}}{\eta} \nabla P \quad (2)$$

where: \vec{v} = Velocity of Flow front

\overline{K} = Permeability tensor

η = Viscosity of the resin

∇P = Pressure gradient (∇ : gradient operator)

Resin infusion and fill time (FT) thus strongly depends upon: Preform Permeability (K) and Resin Viscosity (η). Permeability is a physical property of the fibrous material quantifying the resistance to resin flow. The filling time (FT) and resin infusion flow pattern depend heavily on the preform permeability and is inversely related. Resin viscosity is another key important factor in determining the infusion time. The success of resin infusion depends upon the complete infusion of dry fiber preform prior to resin gelation. Every resin system has a certain gelation time (GT) or "pot life", and it is important to complete resin flow infusion prior to this gelation

time. However, inherent variations and uncertainties in materials and processes affecting the permeability and resin viscosity can lead to changes in resin fill time (FT) and impact the successful infusion prior to resin gelation time. Statistical analysis of such variations and uncertainties in these two key parameters is presented next.

Prior to analysis of coupled parameter variations, a sensitivity analysis was conducted to determine the change in model output values of FT that may result from changes in model input values (for resin viscosity and preform permeability). This sensitivity analysis thus measures the change in the model output in a localized region of the input space. Implicit in this sensitivity analysis were the assumptions that statistical distributions for the input values were correct and that the model is a sufficiently realistic description of the processes taking place in the system.

First step was to examine the effects of changes in a single parameter value or input variable assuming no changes in other inputs. The analysis was then extended to examine the combined effects of multiple, but independent, sources of error (viscosity and permeability). Probability distribution models were developed for each parameter around their fixed mean and standard deviation values. Statistical models employed predict the probability for successful and complete resin infusion before resin gelation defined by $FT \leq GT$. Sensitivity analysis for the parameter uncertainties was developed using the following methodology:

Statistical Analysis Method:

- (1) Generate the parameter space data to accommodate variations in two key parameters: permeability and resin viscosity.

Variable values for permeability (K) and viscosity (η) were generated using normally distributed random errors model around a fixed mean ($\bar{\eta}$ mean resin viscosity; \bar{K} mean preform material permeability) using statistical analysis software. Normal distribution was used to describe the distribution of these errors because it has been observed that normal distribution often describes the actual distribution of the random errors in real-world processes reasonably well. Further, mathematical theory behind the normal distribution is well developed and supports inferences on functions of data and processes being modeled [19,20].

- (2) Perform resin infusion flow modeling simulations for each of the variable values using an in-house deterministic, physic based process flow modeling analysis code. Single parameter models were generated for each input parameter of interest (η , K) and then the model was extended to include both viscosity and permeability, to estimate the corresponding resin infusion time.

- (3) Obtain the probability of success (complete resin infusion of the preform prior to resin gelation). Appropriate cumulative density functions (cdf) were used to calculate the probability that the simulated resin infusion time (FT) based on given permeability and resin viscosity values will be less than the resin gelation time (GT).
- (4) Develop a 95% confidence envelope (bivariate confidence limits) for successful resin infusion – i.e., range of permeability (K) and viscosity (η) values that will result in optimal Fill Time (FT) in more than 95% of cases.

The above two steps (3, 4) were based on the following considerations:

- (a) The process of resin infusion in the preform and the associated infusion time is influenced by statistical (non-deterministic) process uncertainties and variations in key process parameters of resin viscosity and preform permeability. Statistical analysis methodology presented captures these variations and uncertainties. The random errors follow a normal distribution with a mean of zero and a constant standard deviation.
- (b) Data are randomly sampled.

The application of the sensitivity analysis for the uncertainty quantification using statistical probabilistic methodology and computational simulations as discussed above for an illustrative composite helicopter part geometry is presented next.

Sensitivity Analysis for a Complex Helicopter Prototype Part Model Geometry

Figure 2 illustrates the composite part geometry and the associated finite element mesh employed for the deterministic resin infusion process flow modeling. Flow modeling simulations were conducted to find a practical and optimal injection configuration for this complex composite part that would be used in the actual resin infusion of the prototype composite part. Line injection with a resin feed line in the middle of the composite part optimized the FT for this part and provided practical infusion scheme setup for resin infusion [21].

Statistical analysis of variations and uncertainties in two process parameters of resin viscosity and permeability is presented next. A common injection strategy was constantly employed in all cases. Statistical analysis focused on the two key property parameters, resin viscosity and preform permeability, natural variations in which are likely to impact the completion of resin

infusion prior to gelation. Resin viscosity and preform material permeability values employed are those for the fiber and resin used for this helicopter part geometry.

Results and Discussions:

- (1) Parameter space data were generated for variations in these two key parameters, preform permeability and resin viscosity. Means and variations for viscosity and permeability were based on literature review, and ranges that can be expected in this field for these parameters. Statistical analysis software SPSS [22] was used to generate 100 resin viscosity and 100 preform permeability values with the following properties: Both variables were generated with given means, standard deviations and normally distributed random errors as follows: ($\eta \sim N(5.10 \times 10^{-5} \text{ (lbf-s/in}^2\text{)}, 0.28 \times 10^{-5} \text{ (lbf-s/in}^2\text{)}, K \sim N(3.41 \times 10^{-6} \text{ (in}^2\text{)}, 0.59 \times 10^{-6} \text{ (in}^2\text{)})$) as shown in Table 1.

Single parameter Variation:

Single-parameter models for their influence on Fill time (FT) were obtained independently for each of the two variables of interest (keeping the other variable constant). The associated fill time were obtained from process flow modeling simulations for this composite part. It was observed that each of the independent normal random variables η and K were linearly related to the outcome variable FT. Consequently, FT was also a normally distributed random variable for each independent variable η and K .

It was also observed that for each of the FT models mean FT was less than 55 minutes, the gelation time (GT) for the resin used. Resin viscosity values varied within a narrow range and had less impact on fill time with all FT(η) values less than 55min. Preform permeability, K values had a higher range of variations and hence had more impact on FT variability; however it was observed that successful resin infusion ($FT(K) \leq 55\text{mins (GT)}$) occurred in 87% of cases, and for permeability values, $K \geq 2.735 \times 10^{-6} \text{ (in}^2\text{)}$.

Two parameter Variations:

Next, FT was estimated with both parameters varying in the model. From the 100 viscosity values generated, one resin viscosity value was randomly selected from each quintile (20th percentile group), these five viscosity values covered the entire range of the distribution (η : 4.56, 4.73, 5.07, 5.36, and 5.92 ($\times 10^{-5} \text{ lbf-s/in}^2$)). Similarly, 50 values of permeability (K) were also randomly and independently selected from the 100 values generated earlier. Each viscosity value was matched with 50 permeability values creating an input vector of 250 to obtain a

distribution of the associated predicted fill time (FT) employing composite process flow modeling simulations.

The distribution of fill time (FT) values predicted using resin infusion flow modeling was right skewed with mean FT of 49.22 mins and standard deviation of 9.03 mins. Fill time (FT) variable was normalized using a natural log transformation. Both independent random variables (η , K) were related to FT with viscosity, η , explaining approximately 26% ($R^2 = 0.26$), and permeability, K , explaining about 71% ($R^2=0.71$) of the variation in FT. As expected, variations in permeability had more impact on the time required for successful resin infusion than variations in the viscosity.

One of the goals of this study was to develop a bivariate envelope of viscosity and permeability ranges to predict successful resin infusion. A normal cumulative density function (cdf) was generated to obtain the probability of $FT < 55\text{mins}$ (GT) or $\text{LnFT} < 4.0$ for each value of the normal variable LnFT. A plot of cdf of LnFT against permeability for each viscosity value was obtained and studied, and the permeability value corresponding to the 95th percentile of the cdf of LnFT was noted. For example, as shown in figure 4, at viscosity level of $\eta = 4.56 \times 10^{-5}$ (lbf-s/in²), a permeability value, $K=3.58 \times 10^{-6}$ (in²), will give a 95% probability that LnFT will be less than 4.0 (or $FT < 55\text{mins}$ (GT)).

For each of the five viscosity values within its entire range (4.0×10^{-5} lbf-s/in² – 6.0×10^{-5} lbf-s/in²), cdf for the normal random variable LnFT was plotted against permeability to obtain K values for a 95% probability that LnFT will be less than 4.0 or fill time ($FT = 55\text{mins}$), for this composite part geometry and injection conditions. These K values were obtained for each η generating a 95% confidence envelope for successful resin infusion for the given geometry and injection conditions.

The confidence envelope shown in figure 5 provides a tool developed from statistical analysis and simulations that manufacturing engineers can quickly use to predict the potential for successful infusion taking into account variations in two key process parameters of permeability and viscosity. If resin viscosity values and permeability values fall within the range (shaded area – Infusion success chart; figure 5), the probability of successful resin infusion will be at least 95%.

The potential of resin infusion success confidence envelope presented in figure 5 was verified by actual manufacturing of the composite part in our composite processing laboratory. Prior to resin infusion it was observed that resin viscosity value was 5.07×10^{-5} lbf-s/in² and material preform permeability was 3.5×10^{-6} in². Both values are just outside the shaded region of the

confidence predicting a shade less than 95% probability of successful completion. Even with minor variations in the vacuum pressure differential experienced during the actual process, actual resin infusion was completed in 45 minutes. The estimated fill time from the “Probability – LnFillTime” plot obtained during the statistical analysis is about 47 minutes. This matches well the infusion time obtained from an independent process flow modeling simulation performed. Even with minor variations in the vacuum pressure differential experienced during experimental processing, actual resin infusion in the experiments was completed in about 45 minutes. The experimental data and full simulation data clearly conform to confidence envelope predictions for successful resin infusion. They clearly demonstrate the effectiveness of the simulation based statistical analysis of variations and uncertainties in the development of confidence envelope and its application.

The influence of the resin viscosity and permeability on the binary response of successful resin infusion was further statistically analyzed following a logistic regression model [23]. Compared to other regression models, logistic models allow the use of multiple explanatory variables [24] on a binary outcome. Present statistical analysis involves two explanatory variables that pose nonlinearity. Other parameters that influence the successful infusion can also be added to logistic models, and has been employed in the present work.

Logistic Regression Analysis

Logistic regression model is one of the several binary response models that have been used to model dichotomous outcomes in engineering. An interesting case study of this model for failure mode and effect analysis in pharmaceutical tableting tools [25]. Another use of this model in health care six sigma project can be found in [26]. Other examples of this model use for understanding explanatory variables and binary response of space shuttle challenger disaster; incubation temperature and sex (male/female) outcome of turtles hatched can be found in [23].

A logistic regression model containing two continuous independent variables (viscosity and permeability) and a dichotomous dependent variable, FTD (FTD=1 if FT ≤ 55mins (GT) and FTD=0 if FT>55mins (GT).) is considered to predict successful infusion.

General Logistic Regression Model is given by [23,24]:

$$\text{logit}(Y) = \ln\left(\frac{\pi}{1-\pi}\right) = \alpha + \beta_1 X_1 + \beta_2 X_2 \quad (3)$$

Where π = Probability (Y = outcome of interest | $X_1 = x_1, X_2 = x_2$)

$$\pi = \frac{e^{\alpha + \beta_1 X_1 + \beta_2 X_2}}{1 + e^{\alpha + \beta_1 X_1 + \beta_2 X_2}} \quad (4)$$

For our model:

$Y = \text{FTD} = 1 \text{ or } 0$ (if $\text{FT} \leq 55\text{mins (GT)}$ then $\text{FTD}=1$; else if $\text{FT} > 55\text{mins (GT)}$ then $\text{FTD}=0$)

$X_1 = \text{viscosity } (\eta)$, and $X_2 = \text{permeability } (K)$,

The Logistic regression output in Table 3 suggests that the constant α is not a significant part of the model and hence was excluded from the probability analysis. Viscosity and Permeability remain independent and uncorrelated to each other. Probability of success of FTD (π) can also be predicted using:

$$\pi = \frac{e^{-14.44 \eta + 25.76 K}}{1 + e^{-14.44 \eta + 25.76 K}} \quad (5)$$

The coefficients can also be interpreted as follows:

Viscosity: For every 0.1 decrease in Viscosity in the given range, the odds of completing infusion (success) increases 4.2 times. $[(1/10 \times 14.42) = 1.442; e^{(1.442)} = 4.2]$

Permeability: For every 0.1 increase in permeability the odds of completing infusion (success) increases 13.14 times. $[(1/10 \times 25.764) = 2.576; e^{(2.576)} = 13.14]$

As seen earlier, impact of permeability on FT is almost 3 times that of viscosity and logistic regression analysis conforms well to sensitivity analysis inferences.

Concluding Remarks

In engineering applications, due to inherent uncertainty and variation in materials and processes, model uncertainty quantification within an appropriate probabilistic and statistical framework is required. A statistical simulation framework within the context of resin infusion flow modeling was presented and demonstrated in the present paper. Statistical analysis quantified that variations in permeability have higher impact than viscosity variations for successful resin infusion. Even minor variations in the preform layup can lead to notable changes in flow progression and fill time than smaller changes in the resin viscosity. Statistical analysis of uncertainties in parameters and confidence envelope as discussed in the present paper are effective to provide boundaries of actual process parameters for successful resin infusion prior to resin gelation time.

References

1. Mohan R, et al, The Application of Process Simulation Tools to Reduce Risks in Liquid Molding of Composites. *57th American Helicopter Society Forum*, May, 2001.
2. Rudd CD, Long AC, Kendal KN, Mangin CGE. *Liquid Moulding Technologies*: Woodhead Publishing and SAE International, 1997.
3. Mohan RV, Shires DR, Tamma KK, Ngo ND. Flow Channels and Fiber Impregnation Studies for the Process Modeling/Analysis of Complex Engineering Structures Manufactured by Resin Transfer Molding. *Polymer Composites* 1998; **19**(5): 527 – 542.
4. Bickerton S, Mohan RV, Advani SG, Shires DR. Experimental Analysis and Numerical Modeling of Flow Channel Effects in Resin Transfer Molding. *Polymer Composites* 2000; **21**(1).
5. Bruschke M, Advani SG. A Finite Element Control Volume Approach to Mold Filling in Anisotropic Porous Media. *Polymer Composites* 1990; **11** (6).
6. Trouchu F, Gauvin R, Gao D. “Numerical Analysis of the Resin Transfer Molding Process by the Finite Element Method. *Advances in Polymer Technology* 1993; **12**(4): 329-342.
7. Cai Z. Simplified Mold Filling Simulation in Resin Transfer Molding. *J. Composite Materials* 1992, **26**(17): 2606 – 2630.
8. Lin R, Young W, Lee J. Mold Filling and Cure Modeling of RTM and SRIM Processes. *Composites Structures* 1994; **27**: 109 – 120.
9. Mohan RV, Ngo ND, Tamma KK. On a Pure Finite Element Methodology for Resin Transfer Mold Filling Simulations. *Polymer Engineering and Science* 1999; **39**: 26 - 43.
10. Mohan RV et al. Advanced Manufacturing of Large Scale Composite Structures: Process Modeling, Manufacturing Simulations and Massively Parallel Computing Platforms. *Advances in Engineering Software* 1998; **29**: 249 - 264.
11. Mohan RV et al. Three-Dimensional Resin Transfer Molding: Isothermal Process Modeling and Implicit Tracking of Moving Fronts for Thick, Geometrically Complex Composites Manufacturing Applications - Part 2. *Numerical Heat Transfer - Part A, Applications* 1999. **35**: 839 – 858.
12. Young WB. Gate Location Optimization in Liquid Composite Molding using Genetic Algorithms. *J. Composite Materials* 1994; **28**:1098-1113.

13. Mathur R, Advani S, Fink B. Use of Genetic Algorithms to Optimize Gate and Vent Locations for the Resin Transfer Molding Process. *Polymer Composites* 1999; **20**: 167-178.
14. Mathur R, Advani, Fink B. A Sensitivity Based Gate Location Algorithm for Optimal Mold Filling During the Resin Transfer Molding Process. Technical Report 2000. ARL TR-2771, U. S. Army Research Laboratory, Aberdeen Proving Ground, MD.
15. Henz BJ., et al. Process Modeling of Composites by Resin Transfer Molding: Practical Applications of Sensitivity Analysis for Isothermal Considerations. *Int. J. Numerical Methods for Heat and Fluid Flow* 2003; **13**: 415-447.
16. Yu HW, Young WB. Optimal Design of Process Parameters for Resin Transfer Molding. *J. Composite Materials* 1997; **31**: 1113-1140.
17. Luo J. et al. Optimum Tooling Design for Resin Transfer Molding with Virtual Manufacturing and Artificial Intelligence. *Composites: Part A* 2001; **32**: 877-888.
18. Henz, B., Mohan, R., Shires, D. A Hybrid Global-Local Approach for Optimization of Injection Gate Locations in Liquid Composite Molding Process Simulations, *Composites – Part A* 2007; **38**: 1932-1946.
19. Engineering Statistics Handbook: NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook>, April 2011.
20. Devore, J.L.: Probability and Statistics for Engineering and the Sciences, 7th edition, Boston: Brooks/Cole; 2009
21. H. Shiferaw, M.S. Thesis, North Carolina A&T State University, 2011.
22. SPSS: IBM Corp. Released 2010. IBM SPSS Statistics for Windows, Version 19.0. Armonk, NY: IBM Corp.
23. Hosmer DW, Lemeshow S: *Applied Logistic Regression (Wiley Series in Probability and Statistics)*. 3rd edition, New York: Wiley; 2013.
24. D. Cook, P. Dixon, W. M. Duckworth, M. S. Kaiser, K. Koehler, W. Q. Meeker, W. R. Stephenson, Binary Response and Logistic Regression Analysis. Chapter 3, Part of Iowa State University NSF/ILI Project *Beyond Traditional Statistical Methods*, 2001.
25. M. Al-Tahat, A. Jawwad, and Y. Nahleh. Ordinal Logistic Regression Model of Failure Mode and Effects Analysis in Pharmaceutical Tableting Tools. *Engineering Failure Analysis* 2013; **27**: 322-332.

26. F. Meulen, T. Vermaat, and P. Willems. Case Study: An Application of Logistics Regression in a Six Sigma Project in Health Care. *Quality Engineering* 2011; **23**:113-124.

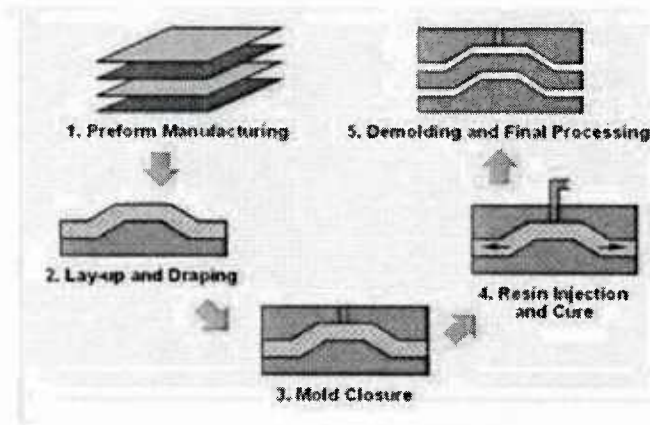


Figure 1: Schematic of Liquid Composite Molding Process for Composite Structures

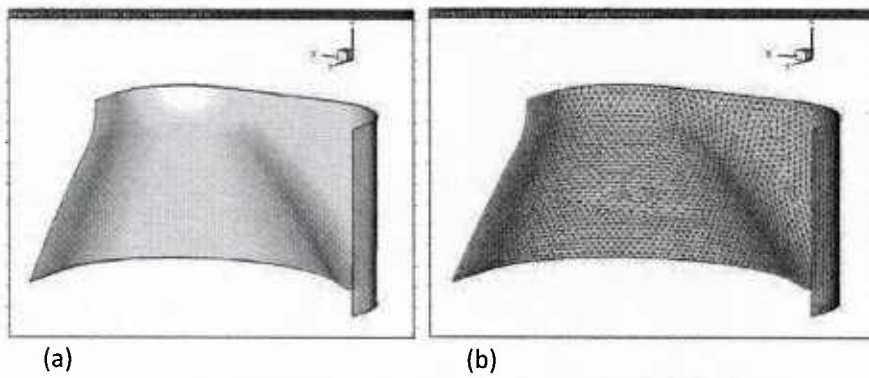


Figure 2: a) composite part geometry. b) Finite element geometry

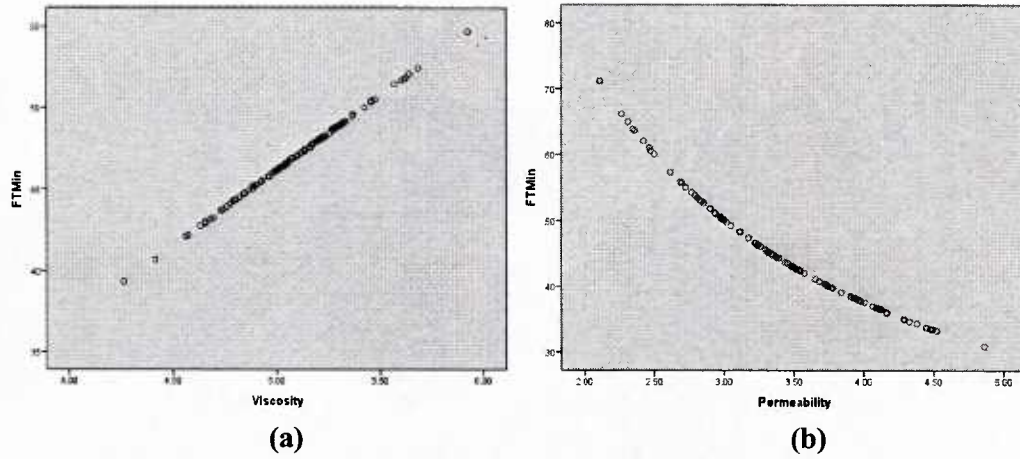


Figure 3: Fill Time was linearly related to (a) Viscosity (directly) and (b) Permeability (inversely)

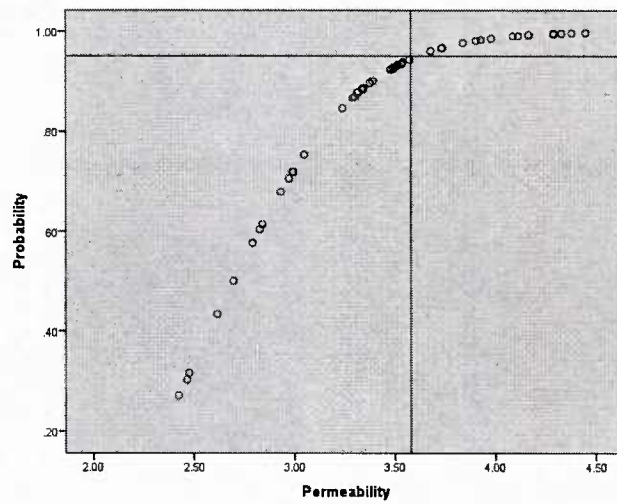


Figure 4: Cumulative Density Function (cdf) of LnFT against Permeability, at viscosity of 4.56×10^{-5} (lbf-s/in²).

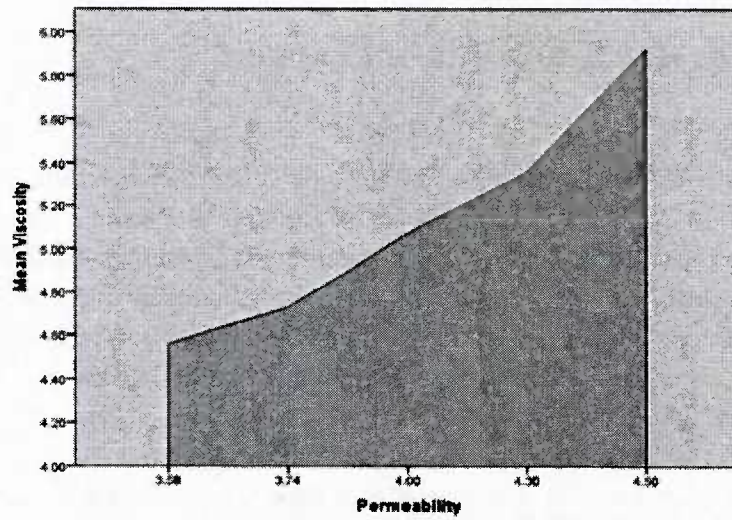


Figure 5: Range of viscosity and permeability values for 95% probability of successful infusion before gelation



C. Educational Activity and Student Support

North Carolina A & T State University (NCAT) is a land grant, historically black college and university (HBCU) with the graduate Masters Computational Science and Engineering (CSE) program established in 2005. The present project efforts were instrumental in the approval of the Ph.D. program in computational science and engineering (CSE) that has admitted its first class in fall 2010. NCAT also recently established the graduate programs (MS and Ph.D.) in nanoengineering. Research initiated and established through the ONR award provided the foundation for this establishment of the new program and paved the way for NCAT to lead such efforts in this arena. NCAT is the first HBCU with a graduate program in nanoengineering. Currently the CSE graduate programs has nearly 25 students and the newly established nanoengineering program has a class of 50 students, majority of whom are under-represented US citizens within the first year of its establishment.

Dr. Mohan is currently a faculty member of nanoengineering focusing on developing the computational nanoengineering area. Within the past two years, Dr. Mohan led the computational nanoengineering focus area efforts in research, education and infrastructure development. The present ONR efforts have enabled these activities and in addition to the research initiatives and activities discussed in this report, Dr. Mohan's efforts had resulted in new research activities focusing on nano to continuum modeling of cementitious materials; computational modeling of bio-nano interfaces; integrated computational materials science and engineering for polymeric composite materials. All these would not have been possible without the ONR funding. The established and growing research in computational nanoengineering is fueling these additional growths. In addition, Dr. Mohan also developed and taught a new graduate course in nanomodeling and applications.

The new research directions and completed research over the years supported through ONR research funding enabled the development and teaching of these new graduate courses. The research areas related to computational nanomechanics, multi-scale modeling and nanoengineered materials leveraged and enabled through the present funding is benefiting the students of this program and will also leverage the opportunities from this new graduate program. The PI and a participating faculty member (Dr. Kelkar) are now part of this new graduate program in nanoengineering. The project funding was utilized to attract qualified minorities to focus in the research areas of computational nanoengineering. The educational activity funds from the project funding provided financial support to several graduate students to obtain graduate education at North Carolina A&T State University. The names and the demographic details of the M.S. and Ph.D. graduate students supported through this project funding are listed below. All these graduate students were either fully or partially supported through the project funding either directly or through the faculty release time funds.

Richard Haney	Ph.D., White, USC, Male (Graduated, Currently at Army Research Laboratory, APG, MD)
Henok Shiferaw	M.S., African American, LPR, Male (Graduated, Currently working for Federal Government)
Elvis Fefey	African American, LPR, Male (Graduated with MS in Computational Science and Engineering; working with Hewlett Packard)
Mariam Sibide	M.S. African American, Female (Graduate with a MS in computational science and engineering and pursuing Ph.D. in computational science and engineering.
Krystal Knight	M.S., African American, USC, Female (Graduated with a MS in computational science and engineering) and employed with industry.
Connie Sidberry	African American, USC, Female (Graduated with MS in Computational Science and Engineering)
Patrick McCarter	African American, USC, Male (Graduated with MS in Computational Science and Engineering and pursuing Ph.D.)
Moussa Seck	African American (Graduated with MS in Computational Science and Engineering and with computing industry)
Bhushan Thatte	Asian, Male (Graduated with MS in Computational Science and Engineering and working in software industry)
Matthew Wiggins	African American, USC, Male (Graduated with MS in Computational Science and Engineering)
Hamed Sibide	African American, Male (Graduated with M.S. in Computational Science and Engineering and working in computing industry)
Abu Rasel	Asian, Male (Graduated with MS in Computational Science and Engineering; currently employed in

	computing industry)
Boding Liu	Asian, Male (Graduated with MS in computational science and engineering)
Terry Corbett	Ph.D., African American, USC, Male
Naveen Chnnannavar	Asian, Male (Graduate with MS in Computational Science and Engineering and employed with industry)
Kristen Rhinehardt	M.S., African American, USC, Female (Graduated with MS in Nanoengineering) (Pursuing Ph.D. in Nanoengineering)
Nafisa Sirelkhatim	Ph.D., African American, USC, Female (Pursuing Ph.D. in Nanoengineering)
Henry Ochije	M.S., African American, Male (Graduated with MS in Nanoengineering, Pursuing Ph.D. in Nanoscience)
Van Nguyen	Ph.D., Asian, USC, Female (Pursuing Ph.D. in Computational Science and Engineering)
Ahmed Mohammed	Ph.D., USC, Male (Graduated with Ph.D. in Computational Science and Engineering and employed as a post-doc)
Mahendran Samykano	Ph.D., Male (Pursuing Ph.D. in Nanoengineering)

The project funding is enabling the education and training of future workforce (esp. underrepresented minorities) in this critical technology area with six Ph.D. students supported through this funding at various levels.

The project efforts enabled the post-doctoral training and mentorship of two researchers (including one female), and a research scientist. Faculty support for the participating faculty was also provided to Dr. Ken Flurchick, and others as part of the project support.

D. Computational and Visualization Hardware / Software

Computational and visualization hardware/software resources are critical components of computational modeling research. The project funding was instrumental in supporting the associated software and system upgrades during the project period benefiting the research and educational needs of the students, faculty and the research focus. In addition, the project funding was instrumental in the award of a NSF major research instrumentation award to acquire a multi-processor SUN Blade system that is currently under installation. The project funding supported the hardware upgrades to the visualization system, new computer hardware and peripherals, software licensing for engineering and visualization analysis software needs of the computational science and engineering graduate program. Some of this software includes ANSYS for finite element analysis, Accelrys and Materials Studio for molecular modeling, COMSOL for multi-physics computational modeling. In addition, the project activities also were instrumental in enabling the analysis capabilities through computational modeling codes such as LAAMPS, GROMACS, etc., that were not regularly employed. The present project efforts have a significant outreach for the research and education in the areas of computational mechanics and materials and enabled the recruitment of under-represented minorities in these critical technical areas. Recently, in August 2014, part of the project funds was utilized to support the purchase of a Cray XC-30 system that would not have been possible without the ONR funding.

